中山大學
SUN YAT-SEN UNIVERSITY

中国科学院高能物理研究所
Institute of High Energy Physics Chinese Academy of Sciences

粒子加速器原理
~大作业讲解~
RCS计算及模拟

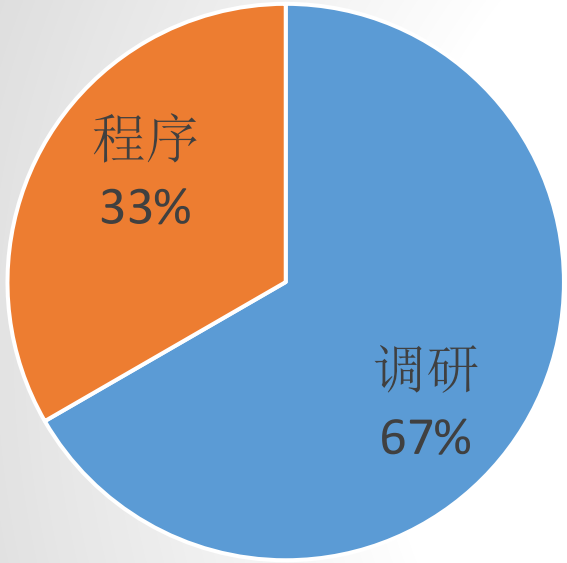
刘星光、苑尧硕
liuxg@ihep.ac.cn

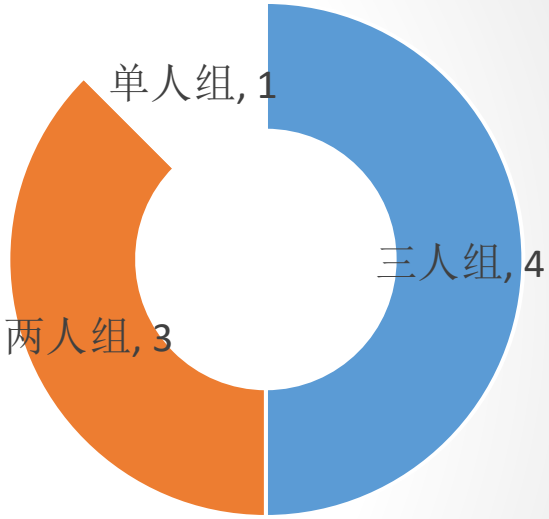
中国科学院高能物理研究所 东莞研究部
2024

能量相关的常用关系变换

In terms of / Wanted	$d\beta$	$d(cp)$	$d\gamma = dE/E_0 = dT/E_0$
$d\beta =$	$d\beta$	$[1 + (cp/E_0)^2]^{-3/2}\, d(cp)/E_0$	$\gamma^{-2}(\gamma^2 - 1)^{-1/2}\, d\gamma$
		$\gamma^{-3}\, d(cp)/E_0$	$\beta^{-1}\, \gamma^{-3}\, d\gamma$
$d(cp) =$	$E_0(1 - \beta^2)^{-3/2}\, d\beta$	$d(cp)$	$E_0\gamma(\gamma^2 - 1)^{-1/2}\, d\gamma$
	$E_0\, \gamma^3\, d\beta$		$E_0\, \beta^{-1}\, d\gamma$
$d\gamma =$ $= dE/E_0 =$ $= dT/E_0 =$	$\beta(1 - \beta^2)^{-3/2}\, d\beta$	$[1 + (E_0/cp)^2]^{-1/2}\, d(cp)/E_0$	$d\gamma$
	$\beta\gamma^3\, d\beta$	$\beta\, d(cp)/E_0$	

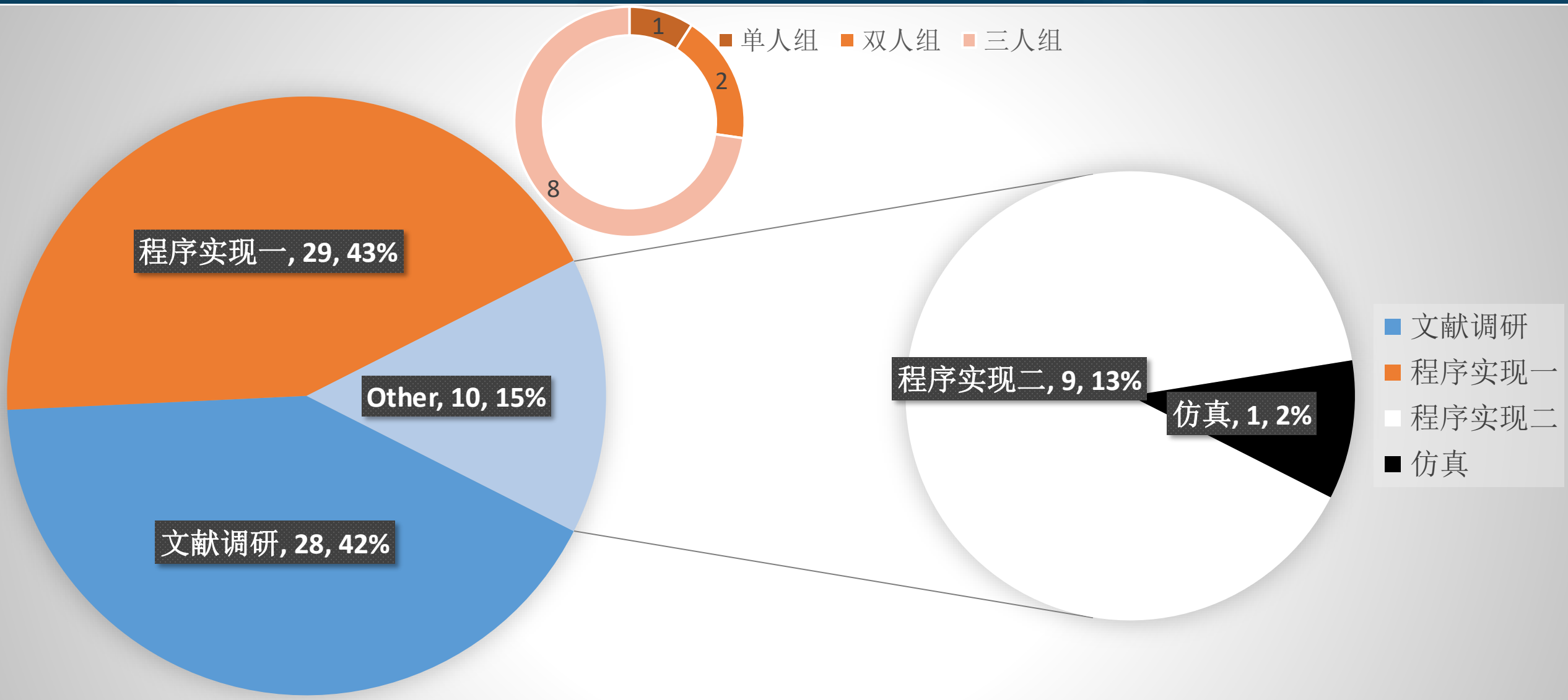
选题情况





合计57人（选课提交54人）

选题情况（2023秋季学期）



作业要求

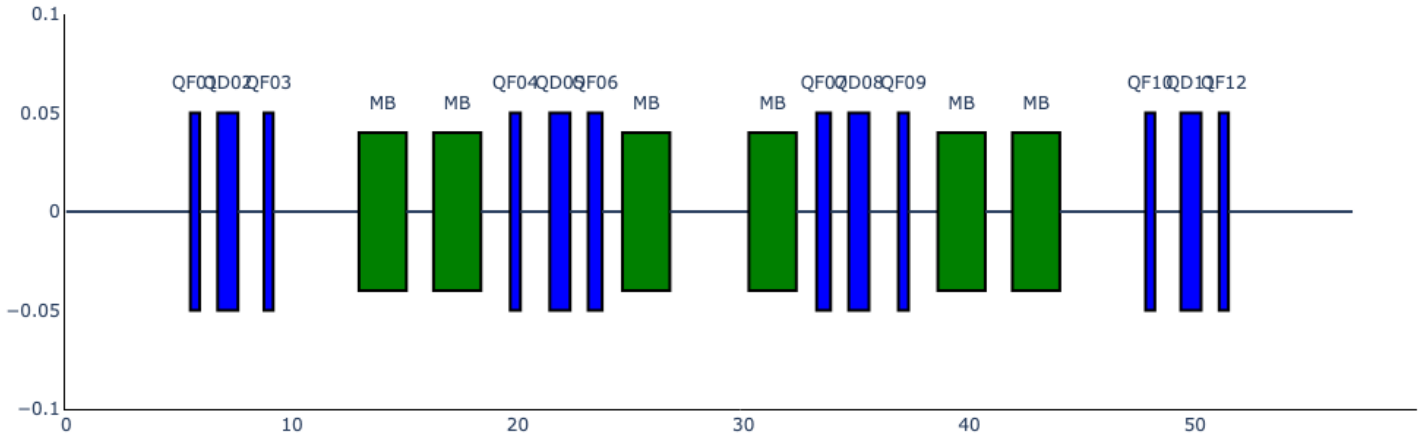
•简介
•数学模型（所用到的公式、坐标及传输矩阵的定义）
•程序实现方法（所使用的语言，类或者函数的定义及程序实现等考虑）
•lattice基本参数计算（CSNS RCS R1）
- 表格形式：元件参数表格
- 以图片格式：β函数等（横轴为s，纵轴包括 β_x ,β_y,α_x,α_y，η_x
  其中η为色散函数，单元节（R1）即可
- 以表格形式：ν_x,ν_y工作点（tune），最大最小β_x ,β_y ，自然色品（chromaticity），穿越能量（\gamma_t)全环周长等

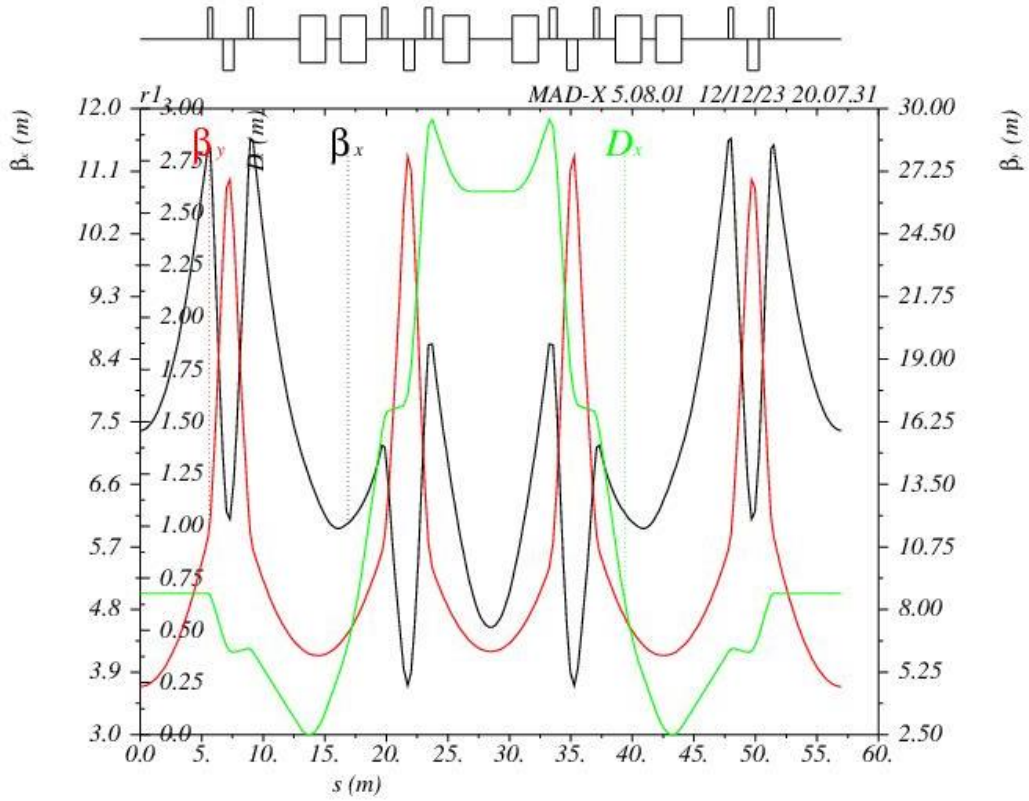
•追踪模拟程序多粒子（基础）
  •选择一个初始粒子（x=0,x'=1,y=1,y'=0，可自行改动）对束流进行跟踪模拟,100圈（或更多)
  •记录粒子在至少五个不同位置（可包含初始位置)所有圈的轨迹，计算该轨迹椭圆的发射度
•追踪模拟程序多粒子（可选）
- 初始分布（可选最简单的均匀分布x-x',y-y'作为初始分布，或选二维高斯分布），粒子数可选10^4~10^6个（图）
- 选择一个初始点（可选择lattice起点）对束流进行跟踪模拟
- 统计逐圈的下述参数（图）变化：束流横向尺寸（\sigma_x,\sigma_y）,束流发射度(\varepsilon_x,\varepsilon_y)
- （生成与lattice匹配的束流分布）

Lattice描述文件

```
// ！ RCS Ring Lattice： R1
// ！ Simplified version， 20231103
// Note that whole RCS ring should have four R1

L01: DRIFT, L = 5.5;
QF01: QUAD, L = 0.41, K1 = 0.74065659951575;
L02: DRIFT, L = 0.800;
QD02: QUAD, L= 0.90, K1 = -0.579364095806417;
L03: DRIFT, L =1.15;
QF03: QUAD, L = 0.41, K1 = 0.664196402392476;
L04: DRIFT, L =3.800;
MB: SBEND, L = 2.1, ANGLE = 0.261799387799149;
L05: DRIFT, L =1.200;
MB: SBEND, L = 2.1, ANGLE = 0.261799387799149;
L106: DRIFT, L = 1.3;
QF04: QUAD, L = 0.45, K1 = 0.570558026371608;
L07: DRIFT, L = 1.3;
QD05: QUAD, L =0.90 , K1 =-0.579364095806417;
L08 :DRIFT, L =0.800;
QF06: QUAD, L = 0.62, K1 = 0.608457760781354;
L09 :DRIFT, L =0.900;
MB: SBEND, L = 2.1, ANGLE = 0.261799387799149;
L10 :DRIFT, L =3.500;
MB: SBEND, L = 2.1, ANGLE = 0.261799387799149;
L11: DRIFT, L = 0.900;
QF07: QUAD, L = 0.62, K1 = 0.608457760781354;
L12 :DRIFT, L =0.800;
QD08: QUAD, L = 0.90, K1 = -0.579364095806417;
L13 :DRIFT, L =1.3;
QF09: QUAD, L = 0.45, K1 = 0.570558026371608;
L14 :DRIFT,L = 1.3;
MB: SBEND, L = 2.1, ANGLE = 0.261799387799149;
L15: DRIFT, L = 1.200;
MB: SBEND, L = 2.1, ANGLE = 0.261799387799149;
L15: DRIFT, L = 3.80;
QF10: QUAD, L = 0.41, K1 = 0.664196402392476;
L16 :DRIFT,L = 1.15;
QD11: QUAD, L= 0.90, K1 = -0.579364095806417;
L17 :DRIFT,L = 0.800;
QF12: QUAD, L = 0.41, K1 = 0.74065659951575;
L18 :DRIFT,L = 5.5;
```

MADX





LENGTH	56.98
ALFA	0.02999595
GAMMATR	5.77389227
Q1	1.32102622
Q2	1.09246772
DQ1	-1.0565439
DQ2	-2.2641779

合抱之木，生于毫末；九层之台，起于垒土；

千里之行，始于足下。

--《老子》第六十四章

All big things start small.

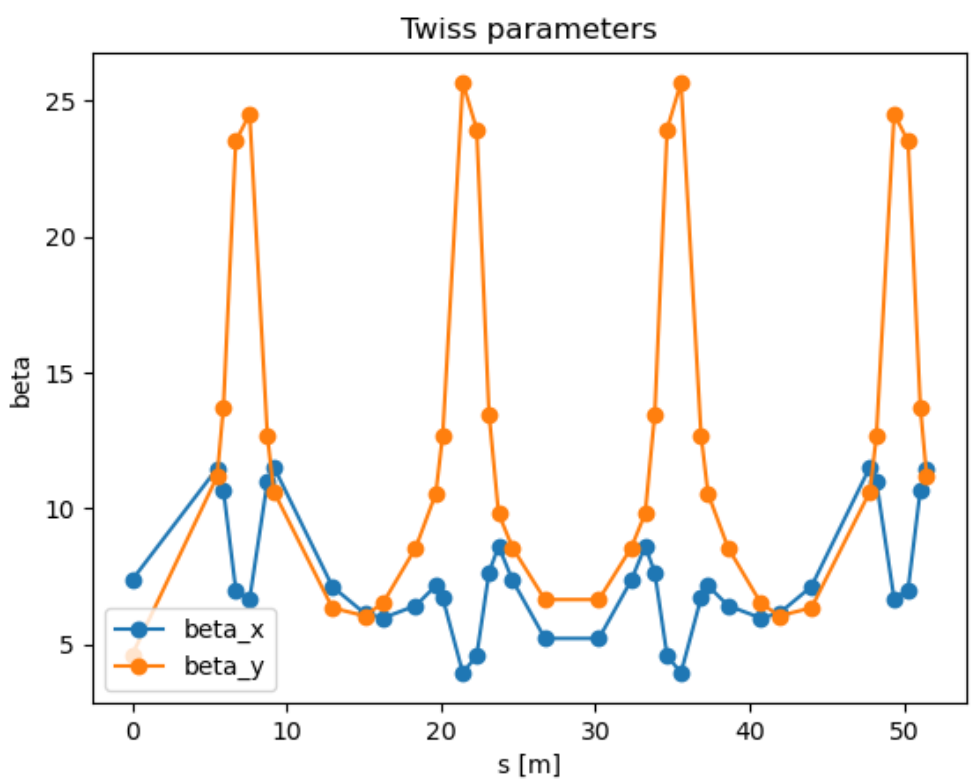
that even the biggest and most ambitious goals can be achieved by taking small, incremental steps.
Gemini Pro (beta)

各元件进出口位置Twiss参数

- 构建基本传输矩阵，根据矩阵性质获得Twiss 参数
- 以各元件为起点构建新的传输矩阵

```
# 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):
    # 储存结果的临时矩阵，先存上第一个
    mt = ml[0]
    # 如果只有一个元件则直接返回
    if len(ml) == 1:
        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2x2的矩阵的迹
    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]
```



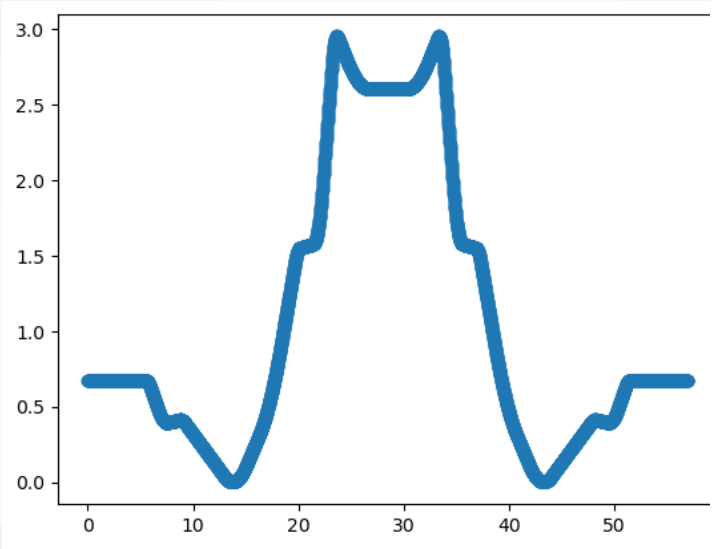
tune x: 0.3210262231879895
tune y: 0.09246771879564876

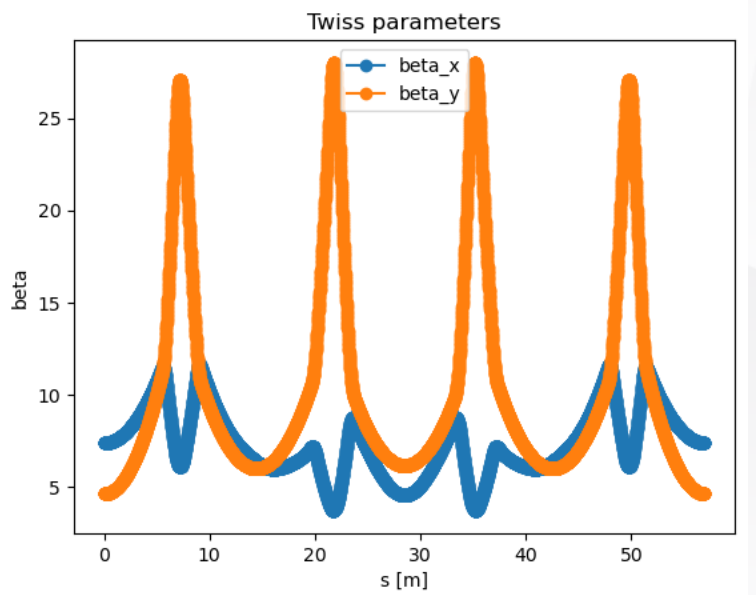
任意位置处的传输矩阵

```
# def a function to return matrix from a lattice list in format of
[type, a1,a2,a3] starting at any give s position
def lat_matrices_s(lat,s):
    mls=[e[1] for e in lat]
    sl=np.cumsum(mls)
    sl_max=sl[-1]
    for i,l in enumerate(mls):
        sl[i]-=l # 修正sl为元件入口处位置
        sl=np.append(sl,sl_max)
     # print(sl)
    id=0
    lower=upper=0.0
    lat_new=[]
    splited=[]
    for i in range(len(sl)-1):
        if sl[i] <= s < sl[i + 1]:
            id=i
            lower=s-sl[i]
            upper=sl[i+1]-s
            if lower >0: # only split when s doesn't coincide with a starting point
                splited=[[lat[id][0],lower,lat[id][2]],[lat[id][0],upper,lat[id][2]]]
                        # if it is sbend whose angle need to be split as well
                        # print("element",lat[id])
                        if lat[id][0]=='SBend':
                                    p=lower/lat[id][1]
                                    a1=lat[id][2]*p
                                    a2=lat[id][2]-a1
                                    splited=[[lat[id][0],lower,a1],[lat[id][0],upper,a2]]
                                    # print("split sbend:",splited)
                        if id==len(lat)-1: # split final element
                                    # print("split final element:",i,lower,upper)
                                    lat_new=lat[:id]+splited
                        else:
                                    lat_new=lat[:id]+splited+lat[id+1:]
            else:
                        lat_new=lat
    # raise an error if s is larger than sl[-1]
    elif s >= sl_max or s<0:
                raise ValueError('s is larger than the last s in the lattice!')
```

```
# loop over s and build the transfer matrix for each s
for i in s:
    [mlx,mly,mls]=lat_matrices_s(rcs_r1,i)
    [b,a,r]=twiss_from_m(Mdot(mlx))
    d,dp=np.linalg.solve(Mdot(mlx)[:2, :2]-np.identity(2),-Mdot(mlx)[0:2, 2].reshape(2, 1))
    twiss_x.append([i,b,a,r,d[0],dp[0]])
    [b,a,r]=twiss_from_m(Mdot(mly))
    twiss_y.append([i,b,a,r])
```

D和D’的计算






```
# if splited is not empty
lat_temp=[]
if splited:
    lat_temp=lat_new[id+1:]+lat_new[:id+1] # staring from s
else:
    lat_temp=lat_new[id:]+lat_new[:id]
# print(lat_temp)
ml=[element_matrix(e[0],*e[1:]) for e in lat_temp]
mlx=list(map(lambda x: x[0],ml)) # select the x matrices
mly=list(map(lambda x: x[1],ml)) # select the y matrices
mls=list(map(lambda x: x[2],ml)) # select the s list
# print(id+1)
return [mlx,mly,mls]
```

α_c, γ_t and chromaticity

```
# calculate the momentum compact factor and chromaticity of the ring:
[mlx,mly,mls]=lat_matrices(rcs_r1) # only mls would be use here
sl=np.cumsum(mls)
sl_max=sl[-1]
for i,l in enumerate(mls):
sl[i]-=l # modify position
print("sl_max:\n",sl_max)

ds=0.01

# set up a s list between 0 and sl[-1], ds as stepsize
sr=np.arange(0,sl_max,ds)

alpha_c=0.0
chromaticity=0.0
chromaticity_y=0.0
rho=2.1/0.261799388
for s in sr:
id=0
for i in range(len(sl)-1):
if sl[i] <= s < sl[i + 1]:
id=i
[mlx,mly,mls]=lat_matrices_s(rcs_r1,s)
[b,a,r]=twiss_from_m(Mdot(mlx))
[by,ay,ry]=twiss_from_m(Mdot(mly))
# if this is a SBend, then acumulate the D(s)/rho
if rcs_r1[id][0] == 'SBend':
d,dp=np.linalg.solve(Mdot(mlx)[:2, :2]-np.identity(2),-Mdot(mlx)[0:2, 2].reshape(2, 1))
alpha_c += d[0]/ rho * ds
chromaticity += 1/(rho*rho) * b * ds
if rcs_r1[id][0] == 'Q':
# print("k value:",s,id,rcs_r1[id][2])
chromaticity += rcs_r1[id][2]*b*ds
chromaticity_y += -rcs_r1[id][2]*by*ds
print("momentum compactor factor is:", alpha_c/sl_max)
print("trasitinoal energy is:", 1/np.sqrt(alpha_c))
print("chromaticity x is:",-chromaticity/(4*np.pi))
print("chromaticity y is:",-chromaticity_y/(4*np.pi))
```

路径长度的不同还可表示为

$$\Delta L = \delta \int \frac{D(s)}{\rho} ds$$

我们可以定义动量压缩因子：

$$\alpha_c = \frac{1}{L_0} \int_0^{L_0} \frac{D(s)}{\rho} ds$$

与γ_t 比较有：$\gamma_t^2 = 1/\alpha_c$

色品：

$$\Delta\nu = \xi\,\delta \qquad \xi = -\frac{1}{4\pi} \oint K(s)\beta(s)\, ds$$

```
sl_max: 56.97999999999998
momentum compactor factor is: 0.02999597661771785
trasitinoal energy is: 5.77388988
chromaticity x is: -1.0880762847134944
chromaticity y is: -2.1734012594276693
```

传输矩阵

直线节

$$m_D = \begin{pmatrix} 1 & l & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

纯二极磁铁

$$m_B = \begin{pmatrix} \mathrm{Cos}\left[\sqrt{K}L\right] & \frac{1}{\sqrt{K}}\mathrm{Sin}\left[\sqrt{K}L\right] & \frac{1}{\sqrt{K}}\left(1-\mathrm{Cos}\left[\sqrt{K}L\right]\right) \\ -\sqrt{K}\mathrm{Sin}\left[\sqrt{K}L\right] & \mathrm{Cos}\left[\sqrt{K}L\right] & \mathrm{Sin}\left[\sqrt{K}L\right] \\ 0 & 0 & 1 \end{pmatrix}$$

组合型二极磁铁（聚焦）

$$m_{BF} = \begin{pmatrix} \mathrm{Cos}\left[\sqrt{K}L\right] & \frac{1}{\sqrt{K}}\mathrm{Sin}\left[\sqrt{K}L\right] & \frac{1}{\rho K}\left(1-\mathrm{Cos}\left[\sqrt{K}L\right]\right) \\ -\sqrt{K}\mathrm{Sin}\left[\sqrt{K}L\right] & \mathrm{Cos}\left[\sqrt{K}L\right] & \frac{1}{\rho\sqrt{K}}\mathrm{Sin}\left[\sqrt{K}L\right] \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x \\ x' \\ \delta \end{pmatrix}_{s2} = \begin{pmatrix} Cos\theta & \rho_0 Sin\theta & \rho_0(1-Cos\theta) \\ -\frac{1}{\rho_0}Sin\theta & Cos\theta & Sin\theta \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ x' \\ \delta \end{pmatrix}_{s1}$$

组合型二极磁铁（散焦）

$$m_{BD} = \begin{pmatrix} \mathrm{Cosh}\left[\sqrt{\mathrm{Abs}[K]}L\right] & \frac{1}{\sqrt{\mathrm{Abs}[K]}}\mathrm{Sinh}\left[\sqrt{\mathrm{Abs}[K]}L\right] & \frac{1}{\rho\mathrm{Abs}[K]}\left(\mathrm{Cosh}\left[\sqrt{\mathrm{Abs}[K]}L\right]-1\right) \\ \sqrt{\mathrm{Abs}[K]}\mathrm{Sinh}\left[\sqrt{\mathrm{Abs}[K]}L\right] & \mathrm{Cosh}\left[\sqrt{\mathrm{Abs}[K]}L\right] & \frac{1}{\rho\sqrt{\mathrm{Abs}[K]}}\mathrm{Sinh}\left[\sqrt{\mathrm{Abs}[K]}L\right] \\ 0 & 0 & 1 \end{pmatrix}$$

Tracking

定义4维矩阵

```
# define the same transfer matrix as we calculate the twiss, ignore the dispersion term

import numpy as np

def D(l):
    return np.array([[1,l,0,0],
                     [0,1,0,0],
                     [0,0,1,l],
                     [0,0,0,1]])

def QF(l,k):
    sk=np.sqrt(k)
    skl=sk*l
    return np.array([[np.cos(skl),np.sin(skl)/sk,0,0],
                     [-sk*np.sin(skl),np.cos(skl),0,0],
                     [0,0,np.cosh(skl),np.sinh(skl)/sk],
                     [0,0,sk*np.sinh(skl),np.cosh(skl)]])

def QD(l,k):
    sk=np.sqrt(np.abs(k))
    skl=sk*l
    return np.array([[np.cosh(skl),np.sinh(skl)/sk,0,0],
                     [sk*np.sinh(skl),np.cosh(skl),0,0],
                     [0,0,np.cos(skl),np.sin(skl)/sk],
                     [0,0,-sk*np.sin(skl),np.cos(skl)]])

# define a function return matrix for sector magnet(recommand rho and angle form):
def SBend(r, a):
    return np.array([[np.cos(a), r*np.sin(a), 0,0],
                     [-np.sin(a)/r, np.cos(a), 0,0],
                     [0,0,1,r*a], # vertical as dirft space
                     [0,0,0,1]])
```

判断元件矩阵及构建元件列表的矩阵

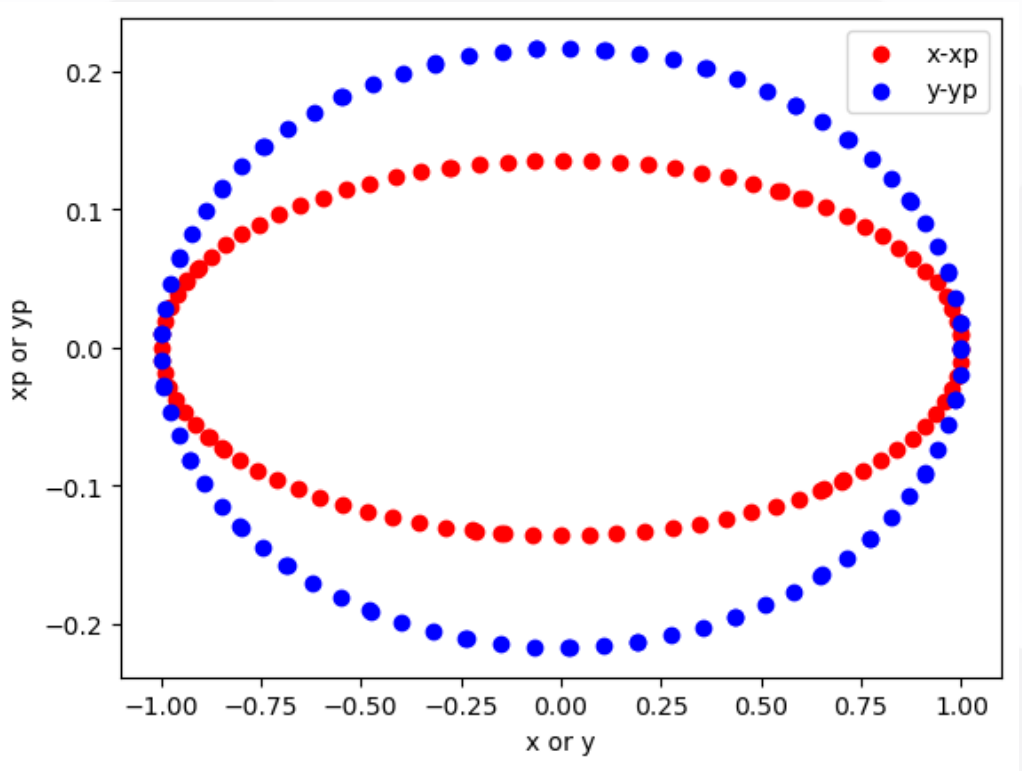
```
# return element matrix by its type and parameters
def element_matrix(*args):
    if args[0] == 'D':
                        l=args[1]
                        return D(l)
    elif args[0] == 'Q':
                        l=args[1]
                        k=args[2]
                        if k>0:
                            return QF(l,k)
                        elif k<0:
                            return QD(l,k)
                        else:
                            return [D(l),D(l),l] # if k=0, return as drift
    elif args[0] == 'SBend':
                        l=args[1]
                        angle=args[2]
                        rho=l/angle
                        return SBend(rho,angle)
    else:
                        print('Unknown element type!')
                        return None

# define a function doing the do production for a list of given matrix:
def Mdot(ml):
    mt = ml[0] # temp matrix to store the results
    if len(ml) == 1: # return if there is only one matrix
        return mt
    else:
        # do the dot production one by one from the 2nd element
        for i in range(1,len(ml)):
            mt = np.dot(ml[i],mt)
    return mt
```

单粒子与多粒子追踪

```
# a simple track function for one particle, p is in the shape of np.array([[x],[xp],[y],[yp]])
def track_single(tm,p):
    return np.dot(tm,p)
# track a bunch of particles
def track(tm,p):
    return np.array([track_single(tm,p[i]) for i in range(len(p))])
```

```
# test particle at position x=1 and y=1, both xp and yp are zero
x0=np.array([1,0,1,0]).reshape(4,1)
# track for 100 times
r=[]
TM=Mdot(ml*4)
r.append(x0) # store the initial distribution first
for i in range(100):
    x0=track_single(TM,x0)
    r.append(x0)
```

单粒子在同一位置的轨迹



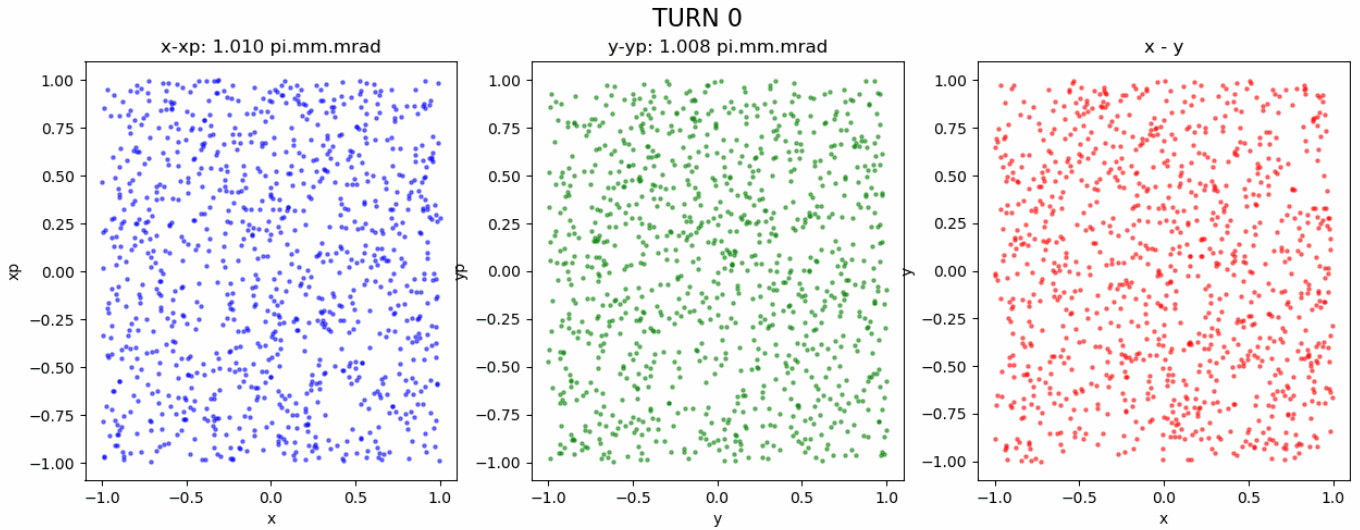
多粒子统计参数计算

```
# calculate the emittance for a given distribution: e=sqrt(mean(x^2)*mean(xp^2)-mean(x*xp))
def distribution_parameters(p):
    ex=np.sqrt(np.mean((p[:,0]-np.mean(p[:,0]))**2)*np.mean((p[:,1]-  np.mean(p[:,1]))**2)-np.mean(p[:,0]*p[:,1])**2)
    ey=np.sqrt(np.mean((p[:,2]-np.mean(p[:,2]))**2)*np.mean((p[:,3]-np.mean(p[:,3]))**2)-np.mean(p[:,2]*p[:,3])**2)
# beam size given as standard deviation
    sx=np.std(p[:,0])
    sy=np.std(p[:,2])
    return ex/np.pi,ey/np.pi,sx,sy
```

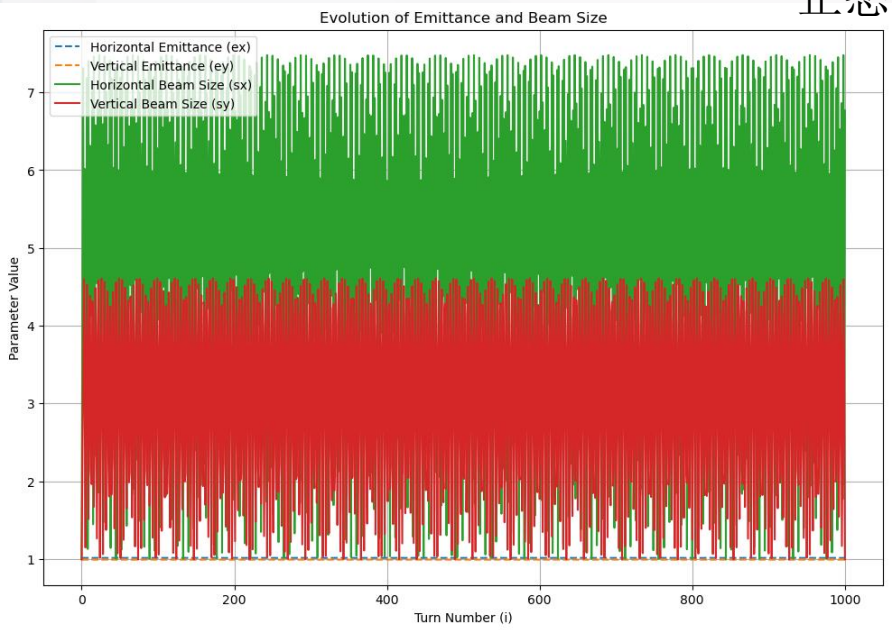
程序追踪大作业展示

均匀分布 vs 正态分布

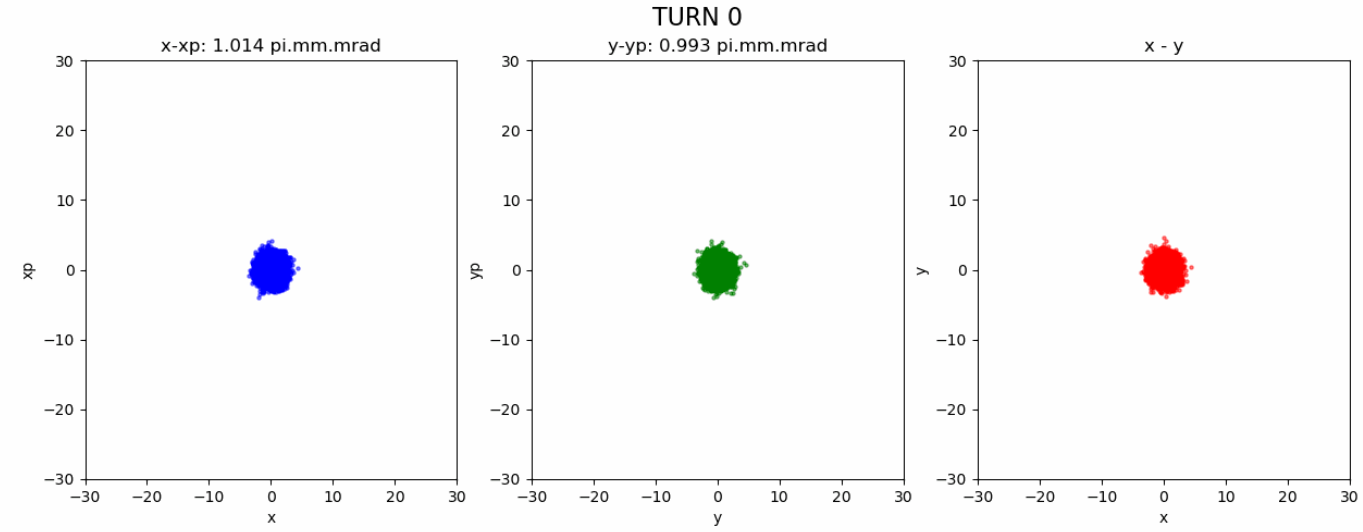
均匀分布



束流参数统计



正态分布



匹配分布

```
# matched beam generation
print("Notice that at the staring point, alpha_x,y is zero, \n which means that the distribution is a typical ellipse.ex=sqrt(sx^2*sxp^2-sxxp^2)")
ex0=ey0=1.0e-6 # pi.mm.mrad
bx0=7.372117913456016
by0=4.612052201102539
sx0=np.sqrt(bx0*ex0*np.pi)
sxp0=np.sqrt(ex0*np.pi/bx0)
# sxp0=ex0/sx0
sy0=np.sqrt(by0*ey0*np.pi)
syp0=np.sqrt(ey0*np.pi/by0)
# syp0=ey0/sy0

print("sx0,sxp0,sy0,syp0",[sx0*1000,sxp0*1000,sy0*1000,syp0*1000])

num_particles =10000
# Covariance matrix
cov_x = [[sx0**2, 0],
[0, sxp0**2]]
cov_y = [[sy0**2, 0],
[0, syp0**2]]

# Generate random samples from a multivariate normal distribution
mean = [0, 0]
x, xp = np.random.multivariate_normal(mean, cov_x, num_particles ).T
y, yp = np.random.multivariate_normal(mean, cov_y, num_particles ).T
```