顧客: 社内	文書種類: 加速器科学
件名: 一般	文書番号: TD-25050B

文書タイトル

ツイスパラメーターとマクロ粒子分布

— 理論的なお話 —

文書発行

(株) オメガソリューションズ　技術部

山本 昌志 (取締役社長)

e-mail: yamamoto.masashi@omega-solutions.co.jp
Tel: 090-9647-0404

発行日 2026.02.15				
確認				作成
検認			照査	
				Masashi Yamamoto

[概要]

本図書は，ツイスパラメーターから粒子分布を作成する方法とプログラムを示す．

[改訂履歴]

TD-25050A	2025.08.30	新規作成
TD-25050B	2026.02.15	プログラム追加等の修正

Omega Solutions

会社名	(株) オメガソリューションズ
web	https://www.omega-solutions.co.jp/
住所	〒135-0043 東京都江東区塩浜 2-7-5-347 クリオレジダンス東京 A–347
e-mail	contact@omega-solutions.co.jp
電話番号	03-6666-5563, 090-9647-0404 (携帯)

コンパイル日時　2026.02.15　01:45

目次

1　はじめに

加速器のビームダイナミクス計算では，適切な初期ビーム分布を設定し，その後の運動を計算する．この初期分布はツイスパラメーターを使い定義することができる．本図書の目的は，ツイスパラメーターから粒子の分布を作成する具体的な方法を示すことである．ただ単に手順を示すのではなく，それに含まれる数学的な背景を示す．

さらに，本図書では理論的背景のみならず，実際の Python プログラムも示す．このプログラムの目的は

$$\alpha = 1.65 \qquad \beta = 0.051 \qquad \varepsilon = 2.894352 \times 10^{-5}$$

のようにツイスパラメーターと RMS エミッタンスを指定すると，図 1 に示す位相空間分布を作成することである．

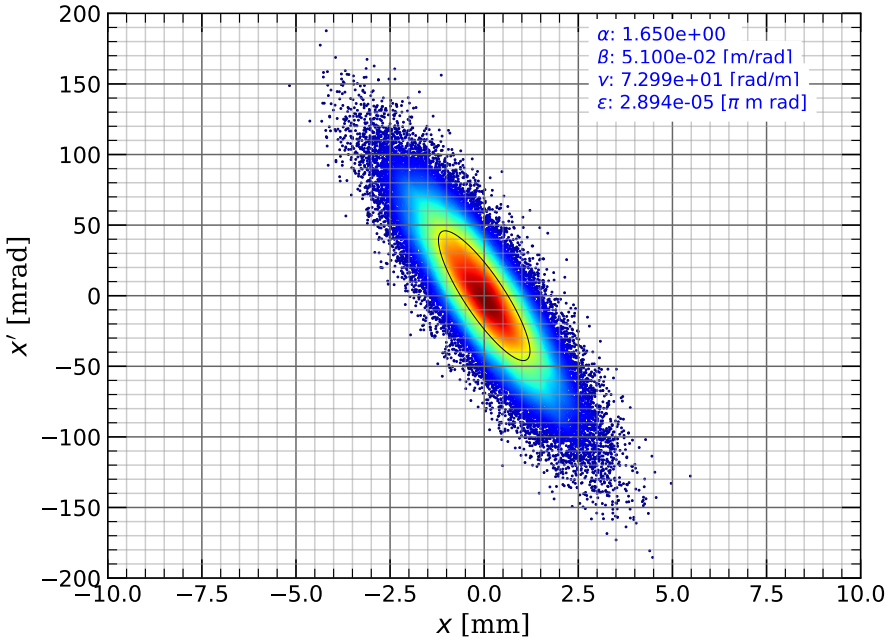


図 1: 位相空間プロット

2 理論

最初に，ツイスパラメーターの物理的な意味を示す．これは，実際に粒子分布を作成する手順を示す 3 節を理解するための基礎知識を与える．

ツイスパラメーターは楕円で表すことができる．この辺りについては，参考文献 [1, 2] で詳細に説明しているので参考になるであろう．

2.1 ツイスパラメーター

粒子の運動は，六次元位相空間 — 例えば $(x,\, y,\, z,\, p_x,\, p_y,\, p_z)$ — で表現できる．粒子はこの位相空間の一点の座標で表すことができる．加速器の世界では，運動量の代わりにビーム軸からの角度 $(x' = p_x/p_z,\, y' = p_y/p_z)$ を使うことが多い．また，加速器のビームは z 軸方向にはバンチが周期的に現れるので，座標と運動量の代わりに位相と平均からの運動エネルギーの差 $(\phi,\, \Delta E)$ で表現する．ϕ は RF の位相，ΔE は粒子の平均からの運動エネルギーの差である．ようするに，加速器科学の世界では位相空間は $(x,\, y,\, \phi,\, x',\, y',\, \Delta E)$ と表現するのである．

6 次元のプロットは描画が不可能なので，通常は (x, x') と (y, y')，$(\phi, \Delta E)$ 三組の二次元位相空間プロットが使われる．位相空間プロットでの粒子の分布の表現には様々な方法がある．例えば，粒子の散布図，密度等高線などである．ツイスパラメーターをプロットする方法もある．以降，ツイスパラメーターでのプロットについて述べる．

ツイスパラメーター $(\alpha,\, \beta,\, \gamma,\, \varepsilon)$ を使い粒子の分布を表す位相空間プロット $(x,\, x')$ は，

$$\begin{bmatrix} x & x' \end{bmatrix} \begin{bmatrix} \gamma & \alpha \\ \alpha & \beta \end{bmatrix} \begin{bmatrix} x \\ x' \end{bmatrix} = I, \qquad I = \varepsilon \tag{1}$$

と表される． I はクーランシュナイダー不変量と呼ばれるもので，後述するがエミッタンスでもある．これが描く曲線はツイスパラメーターに依存し，双曲線や直線，楕円になる [1]．加速器のバンチなので，多くの実務では楕円で近似できる．そのための条件は，

$$0 < \gamma, \qquad 0 < \beta, \qquad 0 < \beta\gamma - \alpha^2 \tag{2}$$

である．その一方，楕円の方程式はパラメーターが三つ (例: 長径，短径，傾き) で表現できる．ツイスパラメーターは 4 個 $(\alpha,\, \beta,\, \gamma,\, \varepsilon)$ あり，明らかにひとつ多い．何らかの拘束条件が必要である．そこで，

$$\beta\gamma - \alpha^2 = 1 \tag{3}$$

とする．こうすることにより，式 (1) の右辺の ε は，楕円の面積を円周率 π で割った値になり，加速器で使うエミッタンスの定義と等しくなる [1]．これで，ツイスパラメーターは便利な表現となるのだ．

具体的な計算は省くが，図 2 に楕円とツイスパラメーターの関係を示す[*1]．

[*1] 具体的な計算が必要な場合は参考文献 [1] に示している．

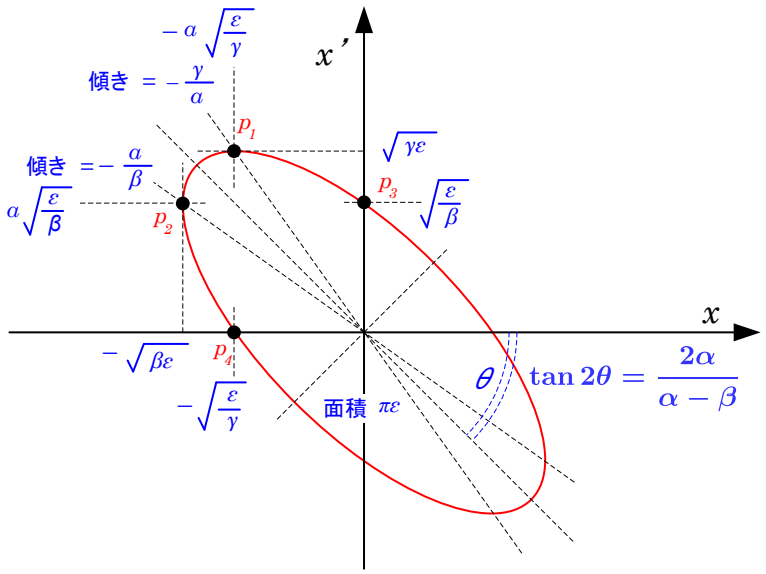


図 2: 楕円とツイスパラメーターの関係

2.2　二次元正規分布

前節では，位相空間の分布は $(x,\, x')$ で表した．ここでは，$(x,\, y)$ で表す．加速器の位相空間に直すときは

$$(x,\, y) \to (x,\, x') \quad \text{or} \quad (y,\, y') \quad \text{or} \quad (\phi,\, \Delta E) \tag{4}$$

と変数変換すればよい．

多次元の正規分布の確率密度関数 f_X は，

$$f_X(x_1, \cdots, x_k) = \frac{\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}} \tag{5}$$

である．$\boldsymbol{\Sigma}$ は一般化された共分散行列で，$\boldsymbol{\Sigma}^{-1}$ はその逆行列，$|\boldsymbol{\Sigma}|$ は行列式: $\det(\Sigma)$ を表す．なお，本図書では，行列式は $|\Sigma|$ で表す．当然のことではあるが，この確率密度関数の x と y の全領域の積分は，1 に規格化されている．ここで，確率変数: $\boldsymbol{x}$ と 平均: $\boldsymbol{\mu}$ は k 次元の列ベクトルである．二次元の共分散行列は，

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} \tag{6}$$

となる．σ_x, σ_y は標準偏差で正の値を持つ．ρ は相関係数で範囲は $-1 \le \rho \le 1$ である．単純化するために平均: $\boldsymbol{\mu} = 0$ とし，式 (6) を式 (5) に代入すると，

$$f_X(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2\sigma_x^2\sigma_y^2(1-\rho^2)} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \sigma_y^2 & -\rho\sigma_x\sigma_y \\ -\rho\sigma_x\sigma_y & \sigma_x^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}\right\} \tag{7}$$

$$= \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma_x^2(1-\rho^2)} & -\frac{\rho}{\sigma_x\sigma_y(1-\rho^2)} \\ -\frac{\rho}{\sigma_x\sigma_y(1-\rho^2)} & \frac{1}{\sigma_y^2(1-\rho^2)} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}\right\} \tag{8}$$

$$= \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)} \left[\frac{x^2}{\sigma_x^2} - \frac{2\rho xy}{\sigma_x\sigma_y} + \frac{y^2}{\sigma_y^2}\right]\right\} \tag{9}$$

となる．この式の指数関数の変数となっている座標 $(x,\, y)$ は，明らかに等密度曲線を表す．例えば，

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \dfrac{1}{\sigma_x^2(1-\rho^2)} & -\dfrac{\rho}{\sigma_x\sigma_y(1-\rho^2)} \\ -\dfrac{\rho}{\sigma_x\sigma_y(1-\rho^2)} & \dfrac{1}{\sigma_y^2(1-\rho^2)} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 1 \tag{10}$$

とすると，これは密度のピークの $\exp(-1/2)$ になる楕円である．左辺の真ん中の行列は $\mathbf{\Sigma}^{-1}$ である．この式の楕円の面積 S は，

$$S = \frac{\pi}{\sqrt{|\mathbf{\Sigma}^{-1}|}} = \pi\sqrt{\det(\mathbf{\Sigma})} = \pi\sigma_x\sigma_y\sqrt{1-\rho^2} = \pi\varepsilon_{rms} \tag{11}$$

となる．ε_{rms} は RMS エミッタンスである[*2]．

2.3 ツイスパラメーターと二次元正規分布の関係

次に，前節で示したツイスパラメーター $(\alpha,\, \beta,\, \gamma, \varepsilon)$ と二次元正規分布のパラメーター $(\sigma_x,\, \sigma_y, \rho)$ の関係を示す．そのため，式 (10) を

$$\begin{bmatrix} x & y \end{bmatrix} \left[\mathbf{\Sigma}^{-1}\right] \begin{bmatrix} x \\ y \end{bmatrix} = 1 \tag{12}$$

に，式 (1) も

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \gamma & \alpha \\ \alpha & \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \varepsilon_{rms} \tag{13}$$

と書きなおす．これらの式から，

$$\mathbf{\Sigma} = \varepsilon_{rms} \begin{bmatrix} \gamma & \alpha \\ \alpha & \beta \end{bmatrix}^{-1} = \varepsilon_{rms} \begin{bmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{bmatrix} \tag{14}$$

で，これを整理すると，

$$\begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} = \varepsilon_{rms} \begin{bmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{bmatrix} \tag{15}$$

となる．ただし，式変形の際に $\beta\gamma - \alpha^2 = 1$ の関係を使った．これでツイスパラメーターの統計的な意味が誰でも分かるのだ．これらの式と二次元の正規分布パラメーターからツイスパラメーターを得る式:

$$\beta = \frac{\sigma_x}{\sigma_y\sqrt{1-\rho^2}} \qquad \gamma = \frac{\sigma_y}{\sigma_x\sqrt{1-\rho^2}} \qquad \alpha = -\frac{\rho}{\sqrt{1-\rho^2}} \qquad \varepsilon_{rms} = \sigma_x\sigma_y\sqrt{1-\rho^2} \tag{16}$$

が得られる．最後の式は式 (11) から得られる．ツイスパラメーターから二次元の正規分布パラメーターへの変換は

$$\sigma_x = \sqrt{\beta\varepsilon_{rms}}, \qquad \sigma_y = \sqrt{\gamma\varepsilon_{rms}}, \qquad \rho = -\frac{\alpha}{\sqrt{\beta\gamma}} \tag{17}$$

となる．

2.4 エミッタンスについて

加速器科学の世界では，困ったことに様々なエミッタンスの定義がある．RMS エミッタンスだけで良さそうであるが，そうもいかない．ここでは，RMS エミッタンスとそのほかのエミッタンスの関係について，簡単にメモしておく．

[*2] 平均 0 の正規分布の場合，二乗平均平方根 (RMS) と標準偏差 (σ) は同じ値になる．

2.4.1 追加説明: RMS エミッタンス

RMS エミッタンスについて，少しだけ注意をしておく．式 (15) の左辺は，統計学の定義により

$$\begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{i=1}^{N} x_i^2 & \frac{1}{N}\sum_{i=1}^{N} x_i y_i \\ \frac{1}{N}\sum_{i=1}^{N} x_i y_i & \frac{1}{N}\sum_{i=1}^{N} y_i^2 \end{bmatrix} = \begin{bmatrix} \langle x^2\rangle & \langle xy\rangle \\ \langle xy\rangle & \langle y^2\rangle \end{bmatrix} = \Sigma \tag{18}$$

となる．ここで，記号 $\langle z\rangle$ は複数のデータ点のリスト z の平均値を表す．これを RMS エミッタンスの式 (16) に代入すると，

$$\varepsilon_{rms} = \sqrt{\langle x^2\rangle\langle y^2\rangle - \langle xy\rangle^2} \tag{19}$$

と見慣れた式が現れる．

2.4.2 その他のエミッタンス

二次元正規分布ではあるが，エミッタンスの定義が RMS と異なる場合がある．例えば「90 % エミッタンス」のような表現である．これは二次元正規分布する全粒子の 90 % の範囲が含まれる領域 (面積を π で割る) をいう．これまでの結果を流用することを考えると，90 % エミッタンスを RMS エミッタンスに変換する必要がある．そのためには，式 (9) を積分する必要がある．このままでは積分は厄介なので，いくつかの変数変換を行う．すると，

$$P_{\eta\sigma} = \int\int_{\eta\sigma} f_X(x,y)\mathrm{d}x\mathrm{d}y = \frac{1}{2\pi\sigma^2}\int_0^{2\pi}\int_0^{\eta\sigma} \exp\left(-\frac{r^2}{2\sigma^2}\right) r dr d\theta = 1 - \exp\left(-\frac{\eta^2}{2}\right) \tag{20}$$

となる．η は RMS エミッタンスの倍数を表す．$\eta = 1$ が RMS エミッタンスで，確率 — 粒子の含まれる割合 — は 0.393 46 である．確率 $P_{\eta\sigma}$ から楕円の右辺の η の計算は，

$$\eta = \sqrt{-2\log\left(1 - P_{\eta\sigma}\right)} \tag{21}$$

とである．90 % エミッタンスは，$\eta = 2.14596$ である．

式 (1) から，η に対応するツイスパラメーターの

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \gamma & \alpha \\ \alpha & \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \eta\varepsilon \tag{22}$$

となり，楕円のプロットが可能である．

あるサンプル点 (x_i, y_i) がツイスパラメーターの楕円の内部/外部の判定が必要な場合がある．その判定は，

$$\begin{bmatrix} x_i & y_i \end{bmatrix} \begin{bmatrix} \gamma & \alpha \\ \alpha & \beta \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \begin{cases} < \eta\varepsilon & \text{楕円内部} \\ = \eta\varepsilon & \text{楕円上} \\ > \eta\varepsilon & \text{楕円外部} \end{cases} \tag{23}$$

で可能である．

3　二次元正規分布の作成方法

ここでは，ツイスパラメーターから二次元正規分布を作成する方法の理論的な背景と具体的なアルゴリズムを示す．具体的な手続きは，

手続き (1) **標準二次元正規分布の生成**　標準偏差: $(\sigma_x = 1, \sigma_y = 1)$，相関係数: $(\rho = 0)$ の二次元のベクトルの集合を (x_i, y_i) を生成する．

手続き (2) **コレスキー分解**　ツイスパラメーターから作られる逆行列をコレスキー分解する．

手続き (3) **座標変換**　コレスキー分解により得られた下三角行列を集合 (x_i, y_i) に乗ずることにより，座標変換を行う．すると，ベクトルの集合 (x'_i, y'_i) が得れれる．

である．これらの手続きにより得られるベクトルの集合 (x'_i, y'_i) がツイスパラメーターが示す二次元の位相空間の分布になる．以降，この手続きが正しい理由を説明する．

3.1　手続き (1) 標準二次元正規分布

標準二次元正規分布 $(\sigma_x = 1, \sigma_y = 1, \rho = 0)$ の共分散行列は，

$$\boldsymbol{\Sigma}_n = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{24}$$

である．この共分散行列が示す分布関数のベクトルの集合 (x_i, y_i) は，Python のようなコンピュータープログラムで容易に生成可能である．これは，RMS エミッタンス: $\varepsilon_n = 1$ の真ん丸の分布である．この分布を

$$\begin{aligned} \boldsymbol{Z} &= \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_N \\ y_1 & y_2 & y_3 & \cdots & y_N \end{bmatrix} \\ &= [\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \cdots, \boldsymbol{z}_N] \end{aligned} \tag{25}$$

と記述する．

3.2　手続き (2) コレスキー分解

式 (12) から，この二次元正規分布は共分散行列で決められることが分かる．この行列は正定値の実数のエルミート行列である．すると，付録 A に示す通りコレスキー分解:

$$\boldsymbol{\Sigma}_{tw} = \varepsilon_{rms} \begin{bmatrix} \gamma & \alpha \\ \alpha & \beta \end{bmatrix}^{-1} = \varepsilon_{rms} \begin{bmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{bmatrix} = \boldsymbol{L}_{tw} \boldsymbol{L}_{tw}^T \tag{26}$$

が可能である．ここで，$\boldsymbol{L}_{tw}$ は下三角行列，$\boldsymbol{L}_{tw}^T$ はその転置行列で上三角行列になる．付録 B に示す通り，この下三角行列は，

$$\boldsymbol{L}_{tw} = \sqrt{\varepsilon} \begin{bmatrix} \sqrt{\beta} & 0 \\ -\dfrac{\alpha}{\sqrt{\beta}} & \sqrt{\gamma - \dfrac{\alpha^2}{\beta}} \end{bmatrix} \tag{27}$$

である．

3.3　手続き (3) 座標変換

標準二次元正規分布:$\boldsymbol{Z}$ の成分をコレスキー分解により得られた下三角行列 $\boldsymbol{L}$ を用いて，座標変換:

$$\boldsymbol{w}_i = \boldsymbol{L}_{tw} \boldsymbol{z}_i \tag{28}$$

を行う．この変換による得られる分布の集合:

$$\begin{aligned}\boldsymbol{W} &= \begin{bmatrix} u_1 & u_2 & u_3 & \dots & u_N \\ v_1 & v_2 & v_3 & \dots & v_N \end{bmatrix} \\ &= [\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3, \cdots, \boldsymbol{w}_N] \end{aligned} \tag{29}$$

とする．これは,

$$\boldsymbol{W} = \boldsymbol{L}_{tw}\boldsymbol{Z} \tag{30}$$

と記述可能である.

以降，この分布が式 (26) の共分散行列になっていることを示す．具体的には

- RMS エミッタンスが ε_{rms} である.
- 共分散行列が $\varepsilon_{rms}\begin{bmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{bmatrix}$ である.

を示す.

RMS エミッタンス

付録 B に示す通り，下三角行列の行列式: $|\boldsymbol{L}_{tw}| = \varepsilon_{rms}$ である．標準二次元正規分布のエミッタンス: $\varepsilon_n = 1$ である．このことから，明らかに分布 $\boldsymbol{W}$ のエミッタンスは，ε_{rms} である.

共分散行列

次に，分布 $\boldsymbol{W}$ の共分散行列を計算する．これは，式 (18) から

$$\begin{aligned}\Sigma_w &= \frac{1}{N}\boldsymbol{W}\boldsymbol{W}^T \\ &= \frac{1}{N}\boldsymbol{L}_{tw}\boldsymbol{z}_i\boldsymbol{z}_i^T\boldsymbol{L}_{tw}^T \\ &= \boldsymbol{L}_{tw}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\boldsymbol{L}_{tw}^T \\ &= \boldsymbol{L}_{tw}\boldsymbol{L}_{tw}^T \\ &= \varepsilon_{rms}\begin{bmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{bmatrix} \end{aligned} \tag{31}$$

となる．これにより，変換された分布 W の共分散行列が目的となってい式 (26) になっていることが分かる.

以上により，ツイスパラメーターから二次元正規分布を得る手順: (1) 標準二次元正規分布，(2) コレスキー分解，(3) 座標変換が正しいことが証明できた.

4　実際のプログラム

4.1　実行プログラムと入力ファイル

これまでに示したツイスパラメーターから実際の分布を作成する Python プログラムと関連ファイルを付録 C に示す．また，表 1 に構成ファイルとその内容を示す．

粒子分布を作成する Python プログラムには，いくつかのライブラリー (モジュール) が使われている．実行する場合は，ライブラリーをインストールした仮想環境を作成することを勧める．

粒子分布の指定方法は入力ファイル: リスト 4，出力の散布図の指定はリスト 3(p.20) のとおりである．それぞれのフォーマットは YAML 形式である．ファイルのコメント欄に，各行の記述内容を示している．

実行方法はバッチファイル: リスト 6(p.24) の通りである．このリストは PowerShell であるが，他のシェルも使用可能である．

表 1: 粒子の分布を作成するプログラム

項目	ファイル名	リスト	内容
Python プログラム	mk_par_dist.py	リスト 2	入力データに従い分布作成を生成
	Scatter.py	リスト 3	散布図の作成
入力データ	例: init_par.yml	リスト 4	粒子分布の指定．ファイル名変更可能.
	例: config.yml	リスト 5	散布図の設定．ファイル名変更可能.
実行バッチファイル	例: run.ps1	リスト 6	実行バッチファイル．ファイル名変更可能.

4.2　出力

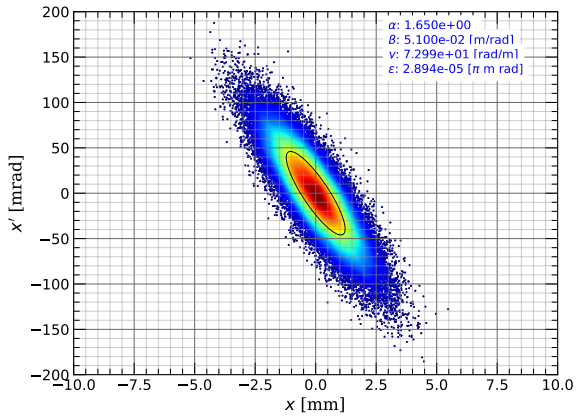
プログラム実行により作成される出力ファイルは,

(1) CST Studio で読み込み可能な PIT 形式の粒子分布を表すテキストファイル
(2) x, y, z の位相空間の散布図 (pdf, png)

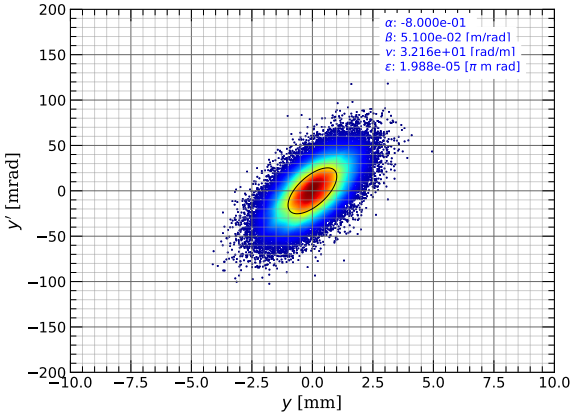
である．PIT 形式のテキストファイルをリスト 1 に示す．このテキストファイルを読み込み，CST Studio で Particle In Cell (PIC) シミュレーションが可能である．また，この PIC 形式ファイルが示す粒子分布を図 3 に示す．

リスト 1: 出力 PIT ファイル (init_par.pit)．粒子の数に応じた行があるので，以下のリストはファイルの一部である．ファイルの数値のフォーマットは.9e であるが，以下は表示のために .3e としている．

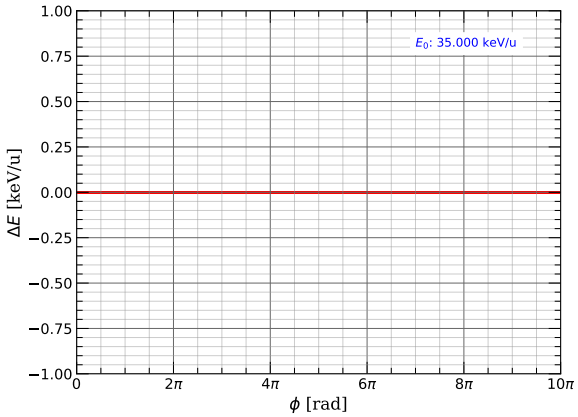
```
% Use always SI units.
% The momentum (mom) is equivalent to beta * gamma.
% The data need not to be chronological ordered.
%
% Columns: pos_x pos_y pos_z mom_x mom_y mom_z mass charge charge(macro) time

 3.702e-04 1.136e-03 0.000e+00 -3.172e-04 2.202e-05 8.631e-03 1.672e-27 1.602e-19 1.000e-15 1.590e-08
 9.118e-04 -3.160e-04 0.000e+00 -6.125e-05 -1.017e-04 8.637e-03 1.673e-27 1.602e-19 1.000e-15 1.571e-08
 -2.370e-03 -1.343e-03 0.000e+00 3.938e-04 -3.145e-04 8.623e-03 1.673e-27 1.602e-19 1.000e-15 2.092e-08
 1.553e-04 -1.108e-03 0.000e+00 -1.084e-04 -3.287e-04 8.631e-03 1.673e-27 1.602e-19 1.000e-15 3.600e-09
 -2.041e-05 -1.154e-04 0.000e+00 -1.698e-04 1.201e-04 8.635e-03 1.673e-27 1.602e-19 1.000e-15 1.791e-08
 1.068e-03 1.893e-03 0.000e+00 -1.384e-04 3.509e-04 8.629e-03 1.673e-27 1.602e-19 1.000e-15 2.073e-08
 8.022e-05 1.696e-03 0.000e+00 2.095e-04 2.726e-05 8.635e-03 1.673e-27 1.602e-19 1.000e-15 9.276e-09
 5.680e-04 1.775e-03 0.000e+00 -3.352e-04 1.864e-04 8.629e-03 1.673e-27 1.602e-19 1.000e-15 1.146e-08
 4.480e-04 2.237e-03 0.000e+00 -3.222e-04 1.377e-04 8.630e-03 1.673e-27 1.602e-19 1.000e-15 1.655e-08
 1.067e-03 1.078e-03 0.000e+00 -3.083e-04 -5.847e-05 8.632e-03 1.673e-27 1.602e-19 1.000e-15 1.316e-08
 -2.246e-04 -1.139e-03 0.000e+00 -7.732e-05 2.263e-04 8.634e-03 1.673e-27 1.602e-19 1.000e-15 2.308e-08
 1.485e-03 -8.617e-04 0.000e+00 -4.463e-04 -7.429e-05 8.626e-03 1.673e-27 1.602e-19 1.000e-15 2.115e-08
```



(a) X 方向の位相空間



(b) Y 方向の位相空間



(c) Z 方向の位相空間

図 3: 本プログラムにより作成された粒子の散布図

付録 A　線形代数のいろいろ

このあたりの話は，線形代数の教科書に書かれている．例えば，参考文献 [3] は良い教科書である．ここに書かれている内容をまとめ，任意の二次元正規分布を作成するための基礎数学を示す．

QR 分解

まずは，行列式がゼロでない行列 $\boldsymbol{A}$ を考える．この場合，QR 分解できる．任意の一次独立な線形ベクトルの集合 $\boldsymbol{A} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \cdots, \boldsymbol{a}_n)$ は，

$$\begin{bmatrix} | & | & | & & | \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \boldsymbol{a}_3 & \cdots & \boldsymbol{a}_n \\ | & | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{q}_1 & \boldsymbol{q}_2 & \boldsymbol{q}_3 & \cdots & \boldsymbol{q}_n \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 & \cdots & \boldsymbol{r}_n \\ 0 & | & | & \cdots & | \\ 0 & 0 & | & \cdots & | \\ \vdots & \vdots & \vdots & \ddots & | \\ 0 & 0 & 0 & \cdots & | \end{bmatrix} \tag{32}$$

と QR 分解できる．右辺の $\boldsymbol{Q} = (\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3, \cdots, \boldsymbol{q}_n)$ はユニタリー (正規直交行列) である．その右辺の上三角行列 $\boldsymbol{R} = (\boldsymbol{r}_1, \boldsymbol{r}_1, \boldsymbol{r}_1, \cdots, \boldsymbol{r}_n)$ は，ベクトルの成分を表す．すなわち，

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \tag{33}$$

である．この式は Gram-Schmidt の直交化である．ここで，$\boldsymbol{R}$ は上三角行列であることに注意．

コレスキー分解

次に，コレスキー分解を示す．次は，条件が厳しい正定値エルミート行列 $\boldsymbol{A}$ を考える．もちろん，この行列の列ベクトルは一次線形独立なので，QR 分解できる．この行列の固有値 $(\lambda_1, \lambda_1, \lambda_1, \cdots, \lambda_n)$ と固有ベクトル $(\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, \cdots, \boldsymbol{s}_n)$ が作る行列をそれぞれ

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \lambda_n \end{bmatrix} \qquad \boldsymbol{S} = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{s}_1 & \boldsymbol{s}_2 & \boldsymbol{s}_3 & \cdots & \boldsymbol{s}_n \\ | & | & | & & | \end{bmatrix} \tag{34}$$

とする．正定値エルミート行列なので，すべての固有値は正の実数で固有ベクトルは直交している．これらを使うと，行列 $\boldsymbol{A}$ は，

$$\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S} = \boldsymbol{\Lambda} \tag{35}$$

と対角化できる．これは，

$$\boldsymbol{A} = \boldsymbol{S}\boldsymbol{\Lambda}\boldsymbol{S}^{-1} \tag{36}$$

$$= \boldsymbol{S}\boldsymbol{\Lambda}^{1/2}\boldsymbol{\Lambda}^{1/2}\boldsymbol{S}^{-1} \tag{37}$$

と変形できる．$\boldsymbol{S}$ は正規化を行い直交行列 (ユニタリー行列) にしても，この式は変わらない．ユニタリー行列の逆行列は，転置行列に等しい．したがって，

$$\boldsymbol{A} = \boldsymbol{S}\boldsymbol{\Lambda}\boldsymbol{S}^{-1} \tag{38}$$

$$= \boldsymbol{S}\boldsymbol{\Lambda}^{1/2}\boldsymbol{\Lambda}^{1/2}\boldsymbol{S}^{T} \tag{39}$$

$$= \left(\boldsymbol{\Lambda}^{1/2}\boldsymbol{S}^{T}\right)^{T}\left(\boldsymbol{\Lambda}^{1/2}\boldsymbol{S}^{T}\right) \tag{40}$$

である．上付きの T は転置行列を表す．ここで，$\boldsymbol{\Lambda}^{1/2}\boldsymbol{S}^T$ は一次独立な線形ベクトルから構成されるので QR 分解が可能である．したがって，

$$\boldsymbol{A} = (\boldsymbol{QR})^T \boldsymbol{QR} \tag{41}$$
$$= \boldsymbol{R}^T \boldsymbol{Q}^T \boldsymbol{QR} \tag{42}$$
$$= \boldsymbol{R}^T \boldsymbol{R} \tag{43}$$
$$= \boldsymbol{LL}^T \tag{44}$$

となる．$\boldsymbol{R}$ は上三角行列なので，その転置行列である $\boldsymbol{L}$ は下三角行列になる．正定値実数エルミート行列 $\boldsymbol{A}$ を $\boldsymbol{LL}^T$ することをコレスキー分解という．複素数の正定値行列の場合は，$\boldsymbol{LL}^\dagger$ となる．ここで，上付きの $\dagger$ は複素共役転置行列を表す．

付録 B　ツイスパラメーターの共分散行列のコレスキー分解

式 (15) ツイスパラメーターを表す共分散行列は

$$\boldsymbol{\Sigma} = \begin{bmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{bmatrix} \tag{45}$$

である．これをコレスキー分解すると下三角行列は,

$$\boldsymbol{L} = \begin{bmatrix} \sqrt{\beta} & 0 \\ -\dfrac{\alpha}{\sqrt{\beta}} & \sqrt{\gamma - \dfrac{\alpha^2}{\beta}} \end{bmatrix} \tag{46}$$

である[*3]．加速器で使われる規約である式 (3) を使うと，明らかに

$$|\boldsymbol{L}| = 1 \tag{47}$$

である．

この結果から,

$$\boldsymbol{\Sigma}_{tw} = \varepsilon_{rms}\boldsymbol{\Sigma} = \varepsilon_{rms}\begin{bmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{bmatrix} \tag{48}$$

のコレスキー分解の下三角行列は,

$$\boldsymbol{L}_{tw} = \sqrt{\varepsilon_{rms}}\begin{bmatrix} \sqrt{\beta} & 0 \\ -\dfrac{\alpha}{\sqrt{\beta}} & \sqrt{\gamma - \dfrac{\alpha^2}{\beta}} \end{bmatrix} \tag{49}$$

であることは明らかである．また，この行列式の値は

$$|\boldsymbol{L}_{tw}| = \varepsilon_{rms} \tag{50}$$

である．

[*3] これは，$\boldsymbol{L} = \begin{bmatrix} \ell_{1\,1} & 0 \\ \ell_{2\,1} & \ell_{2\,2} \end{bmatrix}$ とおいて，$\boldsymbol{\Sigma}_{tp} = \boldsymbol{L}\boldsymbol{L}^T$ を解く

付録 C　ツイスパラメーターから粒子分布作成プログラム

本節では，ツイスパラメーターから実際の粒子の分布を作成プログラムのソードコードを示す．このプログラムは，(1) Python のソースコード，(2) 入力データ，(3) 実行バッチファイルから構成される．

C.1　Python プログラム

Python のプログラムは実際に分布を作成するメインプログラムと散布図を作成するクラスから構成される．

C.1.1　分布作成プログラム

ツイスパラメーターに従った分布を作成するプログラムをリスト 2 に示す．このプログラムは，z 方向に関しては直流 (DC) ビームにも対応している．

リスト 2: 二次元分布作成プログラム (mk_par_dist.py)

```
# -*- coding: utf-8 -*-
"""
mk_par_dist.py
  CST Studio の PIT 形式の粒子分布を作詞えする
    – ツイス > 共分散:
              Sigma = epsiron [[beta, −alpha], [−alph, gamma]]
                    beta gamma − alpha^2 = 1
    – z 方向のビームは「バンチ」と「直流が選択可能」
    – DC の場合は，エネルギーは一定.
    – 出力: CST Studio PIT座標
          ((x, y, z), beta(x, y,z ), gamma, 質量, 電荷, マクロ電荷, 時刻)
"""
import argparse
import os
from pathlib import Path
from typing import Dict, Tuple, List
import matplotlib.pyplot as plt
import numpy as np
import subprocess
import yaml
from Scatter import Scatter as plot

C_LIGHT = 299792458.0  # [m/s]

# ============================================================
# ツイスパラメーターと共分散行列の変換
# ============================================================
def twiss_to_cov(alpha: float, beta: float, epsilon: float) -> np.ndarray:
    gamma = (1.0 + alpha*alpha)/beta
    Sigma = epsilon*np.array([[beta, −alpha],
                              [−alpha, gamma]], dtype=float)
    return Sigma

def cov_to_twiss(Sigma: np.ndarray) -> Tuple[float, float, float, float]:
    eps =   float(np.sqrt(np.linalg.det(Sigma)))
    beta =  float(Sigma[0, 0]/eps)
    gamma = float(Sigma[1, 1]/eps)
    alpha = float(−Sigma[0, 1]/eps)
    return alpha, beta, gamma, eps

# ============================================================
# Sampling
```

```
# ============================================================
def _sample_standard_truncated_2d(N: int, rmax: float,
                                  rng: np.random.Generator)\
                                  -> np.ndarray:
    if rmax <= 0.0:
        raise ValueError("rmax must be positive.")
    out = np.empty((N, 2), dtype=float)
    filled = 0
    batch = max(1024, int(1.2 * N))
    while filled < N:
        z = rng.standard_normal((batch, 2))
        r2 = (z * z).sum(axis=1)
        keep = r2 <= (rmax * rmax)
        k = min(int(keep.sum()), N - filled)
        if k > 0:
            out[filled:filled + k] = z[keep][:k]
            filled += k
    return out


# -------------------------------------
# 2正規分布の作成. D
# コレスキー分解
# 正規分布の生成    (sigma=1)
# 楕円に変換
# -------------------------------------
def sample_plane(alpha: float, beta: float, epsilon: float,
                 N: int, max_sigma: float, rng: np.random.Generator) -> np.ndarray:
    Sigma = twiss_to_cov(alpha, beta, epsilon)
    try:
        L = np.linalg.cholesky(Sigma)
    except np.linalg.LinAlgError:          # 正定値行列から少しずれた場合の処理
        L = np.linalg.cholesky(Sigma + 1e-18 * np.eye(2))

    if max_sigma is None or max_sigma <= 0:
        z = rng.standard_normal((N, 2))
    else:
        z = _sample_standard_truncated_2d(N, max_sigma, rng)
    uv = z @ L.T
    return uv


# -------------------------------------
# マクロ粒子の分布の作成
# -------------------------------------
def make_distributions(par_data: Dict, seed: int = 12345):
    """戻り値
    : (planes, bunch_id)
    planes: dict with keys 'x','y','z', each (N,2) array of (u,v) where
      x->(x,x'), y->(y,y'), z->(phi,dE).
    shape==Bunch:
         total N = particle.N (split across N_bunch), z: Gaussian + phi shift by 2 pi k.
    shape==DC:
         total N = particle.N*N_bunch,  z: phi uniform in [0, 2pi N_bunch), dE = 0
             (E=E0).
    """
    rng         = np.random.default_rng(seed)
    dist        = par_data["distribution"]
    shape       = str(dist["shape"]).strip().upper()
    N_bunch     = int(dist["N_bunch"])
    max_sigma   = float(dist["max_sigma"])
    N           = int(par_data["particle"]["N"])

```

```
    planes: Dict[str, np.ndarray] = {}
    bunch_id: np.ndarray

    N_Bunches = np.full(N_bunch, N)
    # x,y per bunch
    for c in ("x", "y"):
        p = dist[c]
        blocks = []
        for N_par in N_Bunches:
            uv = sample_plane(float(p["alpha"]), float(p["beta"]), float(p["epsilon"]),
                              N=N_par, max_sigma=max_sigma, rng=rng)
            blocks.append(uv)
        planes[c] = np.vstack(blocks)

    if shape == "BUNCH":
        # z per bunch + phi shift
        pz   = dist["z"]
        phi0 = pz['phi0']
        z_blocks = []; bid = []
        for k, N_par in enumerate(N_Bunches):
            uv = sample_plane(float(pz["alpha"]), float(pz["beta"]),
                 float(pz["epsilon"]),
                              N=N_par, max_sigma=max_sigma, rng=rng)
            uv[:,0] += (2.0 * np.pi) * k + np.deg2rad(phi0)
            z_blocks.append(uv); bid.append(np.full(N_par, k, dtype=int))
        planes["z"] = np.vstack(z_blocks)
        bunch_id = np.concatenate(bid) if bid else np.zeros((0,), dtype=int)

    elif shape == "DC":
        # z: φ uniform, dE zeros
        phi = rng.uniform(0.0, 2.0 * np.pi * N_bunch, size=N_Bunches.sum())
        dE  = np.zeros(N_Bunches.sum(), dtype=float)
        planes["z"] = np.column_stack([phi, dE])
        bunch_id = np.floor(phi / (2.0 * np.pi)).astype(int)
    else:
        raise SystemExit(f"distribution.shape は 'Bunch' または 'DC' を指定してください（現在
            : {shape}。}")

    return planes, bunch_id



# ============================================================
# phase space plot
# ============================================================
# --------------------------------------
# Transverse space
#    c: 座標          ('x' or 'y')
#    xxp: 散布図データ        [軸リストx, 軸リストy]
#    dist_cor: ツイスパラメーター, 標準偏差, 相関係数
#    plot_path: プロットのパス
#    BunchNo: バンチの番号   (0, 1, 2, ...)
# --------------------------------------
def _plot_transverse_space(c, xxp, dist_cor, config, plot_path, BunchNo=False):
    ''' shape='DC' の場合の横方向(x, y) の位相空間 '''

    conf = config['plot']['phase_space'][c]
    mm, mrad = 1.0e+3, 1.0e+3
    x, xp   = xxp[0]*mm, xxp[1]*mrad        # [nm] [mrad]
    ux, uxp = dist_cor['av_posi']*mm,     dist_cor['av_angle']*mrad
    sx, sy  = dist_cor['sigma']['u']*mm, dist_cor['sigma']['v']*mrad
    rho     = dist_cor['rho']
    ellipse = {'ux':ux, 'uy':uxp, 'sx':sx, 'sy':sy,'rho':rho, 'eta':1}

```

```
    alpha, beta, gamma = dist_cor['alpha'], dist_cor['beta'], dist_cor['gamma']
    epsilon = dist_cor['epsilon']
    text_label=[
        {'coordinate': [0.65, 0.95], 'label':r'$\alpha$: {0:.3e} '.format(alpha)},
        {'coordinate': [0.65, 0.91], 'label':r'$\beta$: {0:.3e} [m/rad]'.format(beta)},
        {'coordinate': [0.65, 0.87], 'label':r'$\gamma$: {0:.3e} [rad/m]'.format(gamma)},
        {'coordinate': [0.65, 0.83], 'label':r'$\varepsilon$: {0:.3e} [$\pi$ m
            rad]'.format(epsilon)},
    ]
    if BunchNo is not False:
        text_label.append({'coordinate': [0.05, 0.95], 'label':f'Bunch #:
            {BunchNo:02d}'})

    # ------ プロットの作成 ------
    x_axis = {'label': r'${0:s}$ [mm]'.format(c), 'range':conf['position_range']}
    y_axis = {'label': r'${0:s}^\prime$ [mrad]'.format(c),  'range':conf['angle_range']}

    fig = plt.figure(figsize=(8, 6))
    ax1 = fig.add_subplot(1,1,1)


    plot.color_map_with_ellipse(ax1, [x, xp], x_axis, y_axis, ellipse,
                                marker=conf['marker'], size=conf['size'],
                                color_map=conf['color_map'], text=text_label)

    plt.subplots_adjust(left=0.15, bottom=0.2, right=0.85, top=0.9, wspace=0.3,
        hspace=0.3)
    fig.savefig(plot_path+'.pdf', format='pdf', orientation='portrait',
                transparent=False, bbox_inches=None)
    fig.savefig(plot_path+'.png', format='png', orientation='portrait',
                transparent=False, bbox_inches=None)
    fig.clf()
    plt.close('all')

    subprocess.run(["pdfcrop", plot_path+'.pdf', plot_path+'.pdf'], check=True)


# --------------------------------------
# Longitudinal Bunch
# --------------------------------------
def _plot_longitudinal_space(c, dPdEp, dist_cor, config, plot_path, BunchNo=False):
    ''' shape='Bunch' の場合の縦方向(dPhase, dEnergy) の位相空間 '''

    conf = config['plot']['phase_space']['z']
    rad2mdeg = np.rad2deg(1)*1e3
    dP, dE   = dPdE[0]*conf['E_unit']['scale'], xxp[1]*mrad        # [nm] [mrad]
    ux, uxp = dist_cor['av_posi']*mm,     dist_cor['av_angle']*mrad
    sx, sy  = dist_cor['sigma']['u']*mm, dist_cor['sigma']['v']*mrad
    rho     = dist_cor['rho']
    ellipse = {'ux':ux, 'uy':uxp, 'sx':sx, 'sy':sy,'rho':rho, 'eta':1}

    alpha, beta, gamma = dist_cor['alpha'], dist_cor['beta'], dist_cor['gamma']
    epsilon = dist_cor['epsilon']
    text_label=[
        {'coordinate': [0.65, 0.95], 'label':r'$\alpha$: {0:.3e} '.format(alpha)},
        {'coordinate': [0.65, 0.91], 'label':r'$\beta$: {0:.3e} [m/rad]'.format(beta)},
        {'coordinate': [0.65, 0.87], 'label':r'$\gamma$: {0:.3e} [rad/m]'.format(gamma)},
        {'coordinate': [0.65, 0.83], 'label':r'$\varepsilon$: {0:.3e} [$\pi$ mm
            mrad]'.format(epsilon)},
    ]
    if BunchNo is not False:
        text_label.append({'coordinate': [0.05, 0.95], 'label':f'Bunch #:
            {BunchNo:02d}'})
```

```
225
226     # ------ プロットの作成 ------
227     x_axis = {'label': r'${0:s}$ [mm]'.format(c), 'range':conf['position_range']}
228     y_axis = {'label': r'${0:s}^\prime$ [mrad]'.format(c),  'range':conf['angle_range']}
229
230     fig = plt.figure(figsize=(8, 6))
231     ax1 = fig.add_subplot(1,1,1)
232
233
234     plot.color_map_with_ellipse(ax1, [x, xp], x_axis, y_axis, ellipse,
235                                 marker=conf['marker'], size=conf['size'],
236                                 color_map=conf['color_map'], text=text_label)
237
238     plt.subplots_adjust(left=0.15, bottom=0.2, right=0.85, top=0.9, wspace=0.3,
        hspace=0.3)
239     fig.savefig(plot_path+'.pdf', format='pdf', orientation='portrait',
240                 transparent=False, bbox_inches=None)
241     fig.savefig(plot_path+'.png', format='png', orientation='portrait',
242                 transparent=False, bbox_inches=None)
243     fig.clf()
244     plt.close('all')
245
246     subprocess.run(["pdfcrop", plot_path+'.pdf', plot_path+'.pdf'], check=True)
247
248
249 # --------------------------------------
250 # Longitudinal DC
251 # --------------------------------------
252 def _plot_longitudinal_space_DC(dPdE, dist, config, plot_path, BunchNo=False):
253     ''' shape='DC' の場合の横方向(x) の位相空間 '''
254
255     conf = config['plot']['phase_space']['z']
256     dPhi, dE  = dPdE[:,0], dPdE[:,1]*conf['E_unit']['scale']        # 位相, エネルギー
257
258
259     # ------ プロットの作成 ------
260     def fmt_pi(x, pos):
261         k = int(round(x/np.pi))
262         return r"$0$" if k == 0 else rf"${k}\pi$"
263
264     x_axis = {'label': r'$\phi$ [rad]', 'range':[0, 2.0*np.pi*dist['N_bunch']],
265               'ticks_space':2*np.pi, 'ticks_format':fmt_pi}
266     y_axis = {'label': r'$\Delta E$ [{0:s}]'.format(conf['E_unit']['unit']),
          'range':conf['dE_range']}
267
268     fig = plt.figure(figsize=(8, 6))
269     ax1 = fig.add_subplot(1,1,1)
270
271     plot.scatter(ax1, [dPhi, dE], x_axis, y_axis,
272                  marker=conf['marker'], size=conf['size'], color='red',
273                  text=[{'coordinate': [0.7, 0.9], 'label':'$E_0$: {0:.3f}
                         keV/u'.format(dist['z']['E0']*1.0e-3)}])
274
275     # --- ファイル出力 ---
276     plt.subplots_adjust(left=0.15, bottom=0.2, right=0.85, top=0.9, wspace=0.3,
        hspace=0.3)
277     fig.savefig(plot_path+'.pdf', format='pdf', orientation='portrait',
278                 transparent=False, bbox_inches=None)
279     fig.savefig(plot_path+'.png', format='png', orientation='portrait',
280                 transparent=False, bbox_inches=None)
281     fig.clf()
282     plt.close('all')
283
```

```
284     subprocess.run(["pdfcrop", plot_path+'.pdf', plot_path+'.pdf'], check=True)
285
286 # ------------------------------------
287 # plot phase space control
288 # ------------------------------------
289 def plot_phase_space(planes, par_data, out_path, config):
290
291     os.makedirs(os.path.join(os.path.dirname(out_path), 'plot'), exist_ok=True)
292
293     N_par_bunch = par_data['particle']['N']
294     if par_data['distribution']['shape'].upper() == 'DC':
295         # --- 横方向 (x, y) ---
296         for c in ('x', 'y'):
297             x_axis, y_axis=planes[c][:,0], planes[c][:,1]
298             plot_path = Path(os.path.join(os.path.dirname(out_path), 'plot',
                    c)).as_posix()
299             _plot_transverse_space(c, [x_axis, y_axis], par_data['distribution'][c],
                    config, plot_path)
300         # --- 縦方向 (z) ---
301         plot_path = Path(os.path.join(os.path.dirname(out_path), 'plot', 'z')).as_posix()
302         _plot_longitudinal_space_DC(planes['z'], par_data['distribution'], config,
                plot_path)
303
304     elif par_data['distribution']['shape'].upper() == 'BUNCH':
305         # --- 横方向 (x, y) ---
306         for ib in range(par_data['distribution']['N_bunch']):
307             start, stop = ib*N_par_bunch, (ib+1)*N_par_bunch
308             for c in ('x', 'y'):
309                 plot_path = Path(os.path.join(os.path.dirname(out_path), 'plot',
                        f'{ib:02d}_{c}')).as_posix()
310                 x_axis, y_axis=planes[c][:,0][start:stop], planes[c][:,1][start:stop]
311                 _plot_transverse_space(c, [x_axis, y_axis], par_data['distribution'][c],
312                                        config, plot_path, BunchNo=ib)
313     else:
314         print('\n\nError !! Setting (distribution > shape) should be DC or Bunch.')
315         print(f'your setting is {par_data["distribution"]["shape"]}.')
316         exit()
317
318
319 # ============================================================
320 # I/O
321 # ============================================================
322 # ------------------------------------
323 # インプットファイルの読み込みと設定
324 # ------------------------------------
325 def set_parameters(in_file: str, config_file: str) -> Tuple[Dict, Dict, Dict]:
326     ''' インプットファイルの読み込みと設定 '''
327     with open(in_file, "r", encoding="utf-8") as f:
328         input_data = yaml.safe_load(f)
329     par_data = dict(input_data)
330
331     for c in ("x", "y", "z"):
332         p = par_data["distribution"][c]
333         alpha = float(p["alpha"]); beta = float(p["beta"]); eps = float(p["epsilon"])
334         gamma = (1.0 + alpha * alpha) / beta
335         p["gamma"] = gamma
336         p["sigma"] = {"u": np.sqrt(beta * eps), "v": np.sqrt(gamma * eps)}
337         p["rho"] = -alpha / np.sqrt(1.0 + alpha * alpha)
338
339     with open(config_file, "r", encoding="utf-8") as f:
340         read_data = yaml.safe_load(f)
341     config_data = dict(read_data)
342
```

```
343
344     return input_data, par_data, config_data
345
346
347 # ============================================================
348 # PIT writer
349 # ============================================================
350 # --------------------------------------
351 # calculate beta * gamma
352 #         xp, yp: 角度  [deg]
353 #         E: エレメンタリー粒子の運動エネルギー        [J]
354 # --------------------------------------
355 def betagamma_components(xp: np.ndarray, yp: np.ndarray, E: np.ndarray, mass_kg: float):
356     mc2 = mass_kg * C_LIGHT**2                          # mc^2
357     G        = 1.0 + (E/mc2)                            # gamma
358     BG       = np.sqrt(G*G - 1.0)                       # beta * gamma
359     B_par_Bz = np.sqrt(1.0 + xp*xp + yp*yp)             # beta/beta_z
360     BzG      = BG / B_par_Bz                            # beta_z * gamma
361     BxG      = xp * BzG                                 # beta_x * gamma
362     ByG      = yp * BzG                                 # beta_y * gamma
363
364     return BxG, ByG, BzG
365
366
367 # --------------------------------------
368 # make pit file which is CST Studio PIC input
369 # --------------------------------------
370 def write_output(planes: Dict[str, np.ndarray], par_data: Dict, out_path: str, bunch_id:
        np.ndarray) -> None:
371     """
372     PIT format (text):
373     Columns: pos_x pos_y pos_z mom_x mom_y mom_z mass charge charge(macro) time
374     mom_* = beta*gamma components. Units: SI. pos_z=z0 (fixed). time = phi/(2*pi*RF).
375     All numeric fields are formatted as lowercase scientific with 9 decimals: '%.9e'.
376     """
377
378     acc          = par_data['accelerator']
379     par          = par_data['particle']
380     dist         = par_data['distribution']
381
382     N_par_bunch  = par_data['particle']['N']
383     N_par_total  = planes['z'].shape[0]
384
385     X, Xp   = planes['x'][:,0], planes['x'][:,1]        # [m] [rad]
386     Y, Yp   = planes['y'][:,0], planes['y'][:,1]        # [m] [rad]
387     Phi, dE = planes['z'][:,0], planes['z'][:,1]        # [rad] [eV]
388     Z = np.full_like(X, dist['z']['z0'], dtype=float)  # pos_z = z0 (fixed)
389
390     E0     = float(dist['z']['E0'])                     # [eV]
391     RF     = float(acc['RF'])                           # [Hz]
392     Ib     = float(dist['Ib'])                          # [A]
393     mass   = float(par['m'])                            # [kg]
394     charge = float(par['e'])                            # [C]
395
396     t = Phi/(2.0*np.pi*RF)                              # time from phi
397     E = E0 + dE                                         # Kinetic energy [eV]
398     bgx, bgy, bgz = betagamma_components(Xp, Yp, E*np.abs(charge), mass)  # beta * gamma
399     q_macro_charge = Ib / (RF * N_par_bunch)            # Charge per macro paricle q =
            Ib/(N*RF)
400     q = np.full(N_par_total, q_macro_charge, dtype=float)
401
402     # Formatter
403     def fe9(v: float) -> str:
```

```
404         try:
405             if np.isfinite(v):
406                 return '{0:.9e}'.format(v)
407             else:
408                 return "nan"
409         except Exception:
410             return "nan"
411 
412     os.makedirs(os.path.dirname(out_path) or ".", exist_ok=True)
413     with open(out_path, "w", encoding="utf-8") as f:
414         f.write("% Use always SI units.\n")
415         f.write("% The momentum (mom) is equivalent to beta * gamma.\n")
416         f.write("% The data need not to be chronological ordered.\n")
417         f.write("%\n")
418         f.write("% Columns: pos_x pos_y pos_z mom_x mom_y mom_z mass charge
                charge(macro) time\n\n")
419         for i in range(N_par_total):
420             row = " ".join([
421                 fe9(X[i]), fe9(Y[i]), fe9(Z[i]),
422                 fe9(bgx[i]), fe9(bgy[i]), fe9(bgz[i]),
423                 fe9(mass), fe9(charge), fe9(q[i]), fe9(t[i])
424             ])
425             f.write(row + "\n")
426 
427 
428 # ============================================================
429 # メイン関数
430 # ============================================================
431 def main():
432     import argparse
433     ap = argparse.ArgumentParser(description="初期粒子分布の作成（CST PIT 出力）")
434     ap.add_argument("-i", "--input", required=True, help="インプットYAML")
435     ap.add_argument("-o", "--output", required=True, help="出力ファイル（PIT.）pit")
436     ap.add_argument("--seed", type=int, default=12345, help="乱数シード 再現用()")
437     ap.add_argument("--config", required=True, help="設定ファイル")
438     args = ap.parse_args()
439 
440     input_data, par_data, config_data = set_parameters(args.input, args.config)
441     planes, bunch_id = make_distributions(par_data, seed=args.seed)
442     write_output(planes, par_data, args.output, bunch_id)
443     plot_phase_space(planes, par_data, args.output, config_data)
444 
445     print("N =", planes['z'].shape[0], "shape =", par_data['distribution'].get('shape'))
446     print(f"Saved PIT: {args.output}")
447 
448 if __name__ == "__main__":
449     main()
```

C.1.2 散布図作成クラス

リスト 3 は，作成された粒子の分布をプロットするクラスである．これは，メインプログラムのリスト 2 から呼び出される．

リスト 3: 散布図作成クラス (Scatter.py)

```
1 # -*- coding:utf-8 -*-
2 import matplotlib.ticker as ticker
3 import mpl_toolkits.axes_grid1
4 import numpy as np
5 from scipy.stats import gaussian_kde
6 
7 class Scatter():
```

```
8       ''' 二次元散布図の作成 '''
9
10      # ============================================================================
11      # 散布図
12      # ============================================================================
13      @classmethod
14      def scatter(cls, axis, xy, x_axis, y_axis, **kwargs):
15          ''' 散布図を作成する '''
16
17          for i, txt in enumerate(kwargs['text']):
18              rot = txt['rotation'] if 'rotation' in txt.keys() else 0.0
19              fs  = txt['fontsize'] if 'fontsize' in txt.keys() else 10
20              fc  = txt['color']    if 'color'    in txt.keys() else 'blue'
21              axis.text(txt['coordinate'][0], txt['coordinate'][1], txt['label'],
22                        rotation=rot, fontsize=fs, color=fc, transform=axis.transAxes,
23                        backgroundcolor="white")
24
25          axis.set_xlabel(x_axis['label'], fontsize=14, fontname='serif',
                fontweight='normal')
26          axis.set_ylabel(y_axis['label'], fontsize=14, fontname='serif',
                fontweight='normal')
27          axis.tick_params(direction='in', axis='both', length=10, which='major',\
28                           labelsize=12,\
29                           bottom=True, top=True, left=True, right=True)
30          axis.tick_params(direction='in', axis='both', length=5,  which='minor',\
31                           bottom=True, top=True, left=True, right=True)
32          if 'ticks_space' in x_axis.keys():
33              axis.xaxis.set_major_locator(ticker.MultipleLocator(x_axis['ticks_space']))
34          if 'ticks_format' in x_axis.keys():
35                  axis.xaxis.set_major_formatter(ticker.FuncFormatter(x_axis['ticks_format']))
36          if 'ticks_space' in y_axis.keys():
37              axis.yaxis.set_major_locator(ticker.MultipleLocator(y_axis['ticks_space']))
38          if 'ticks_format' in y_axis.keys():
39              axis.yaxis.set_major_formatter(ticker.FuncFormatter(y_axis['ticks_format']))
40          if isinstance(x_axis['range'], (list, np.ndarray)):
41              axis.set_xlim([x_axis['range'][0], x_axis['range'][1]])
42          if isinstance(y_axis['range'], (list, np.ndarray)):
43              axis.set_ylim([y_axis['range'][0], y_axis['range'][1]])
44          axis.minorticks_on()
45          axis.grid(which='major', color='#666666', linestyle='-')
46          axis.grid(which='minor', color='#999999', linestyle='-', alpha=0.5)
47
48          axis.scatter(xy[0], xy[1], marker=kwargs['marker'], s=kwargs['size'],
49                       c=kwargs['color'], linewidth=0.2)
50
51          return axis
52
53
54      # ============================================================================
55      # カラーマップと楕円
56      #    gamma*(x-ux)**2 + 2*alpha*(x-ux)*(y-uy) + beta*(y-uy)**2 = eps
57      #    ellips = [ux, uy, alpha, beta, eps]
58      # ============================================================================
59      @classmethod
60      def color_map_with_ellipse(cls, axis, xy, x_axis, y_axis,  ellipse, **kwargs):
61          ''' カラーマッププロットを作成する '''
62          if not 'linewidth' in kwargs.keys(): kwargs['linewidth'] = 0.2
63
64
65          for i, txt in enumerate(kwargs['text']):
66              rot = txt['rotation'] if 'rotation' in txt.keys() else 0.0
67              fs  = txt['fontsize'] if 'fontsize' in txt.keys() else 10
68              fc  = txt['color']    if 'color'    in txt.keys() else 'blue'
```

```
69              axis.text(txt['coordinate'][0], txt['coordinate'][1], txt['label'],
70                        rotation=rot, fontsize=fs, color=fc, transform=axis.transAxes,
71                        backgroundcolor="white")
72          axis.set_xlabel(x_axis['label'], fontsize=14, fontname='serif',
                fontweight='normal')
73          axis.set_ylabel(y_axis['label'], fontsize=14, fontname='serif',
                fontweight='normal')
74          axis.tick_params(direction='in', axis='both', length=10, which='major',\
75                           labelsize=12,\
76                           bottom=True, top=True, left=True, right=True)
77          axis.tick_params(direction='in', axis='both', length=5,  which='minor',\
78                           bottom=True, top=True, left=True, right=True)
79          if isinstance(x_axis['range'], (list, np.ndarray)):
80              axis.set_xlim([x_axis['range'][0], x_axis['range'][1]])
81          if isinstance(y_axis['range'], (list, np.ndarray)):
82              axis.set_ylim([y_axis['range'][0], y_axis['range'][1]])
83          axis.minorticks_on()
84          axis.grid(which='major', color='#666666', linestyle='-')
85          axis.grid(which='minor', color='#999999', linestyle='-', alpha=0.5)
86
87          # ------ 散布図の作成 ------
88          x, y = xy[0], xy[1]
89          xy_vstack = np.vstack([x,y])
90          xy    = np.vstack([x,y])
91          dens = gaussian_kde(xy)(xy)
92          idx = dens.argsort()
93          xx, yy, ddens = x[idx], y[idx], dens[idx]
94
95          axis.scatter(xx, yy, marker=kwargs['marker'], s=kwargs['size'],\
96                       c=ddens, cmap=kwargs['color_map'], linewidth=kwargs['linewidth'])
97
98          # ------ 楕円の作成 ------
99          if ellipse:
100             ux  = ellipse['ux']
101             uy  = ellipse['uy']
102             sx  = ellipse['sx']
103             sy  = ellipse['sy']
104             rho = ellipse['rho']
105             eta = ellipse['eta']
106
107             S = eta*np.pi*sx*sy*np.sqrt(1-rho**2)
108             print('section of ellipse: {0:.3f}'.format(S))
109
110             cov =[[sx**2, rho*sx*sy],[rho*sx*sy, sy**2]]
111             Sig_inv = np.matrix(np.linalg.inv(cov))
112
113             #  --- 楕円内のポイントのカウント ---
114             n_in = 0
115             for XX, YY in zip(x, y):
116                 dis = np.matrix([XX-ux, YY-uy])@Sig_inv@np.matrix([XX-ux, YY-uy]).T
117                 if dis[0,0] < eta: n_in += 1
118             print('Number of Ellips: {0:d}'.format(n_in))
119             gamma = Sig_inv[0, 0]
120             beta  = Sig_inv[1, 1]
121             alpha = Sig_inv[0, 1]
122
123             mg_x, mg_y = np.mgrid[x_axis['range'][0]: x_axis['range'][1]: 4001j,
124                                   y_axis['range'][0]: y_axis['range'][1]: 4001j]
125
126             axis.contour(mg_x, mg_y,
127                          gamma*(mg_x-ux)**2+2*alpha*(mg_x-ux)*(mg_y-uy)+beta*(mg_y-uy)**2,
                                 [eta],
128                          colors=['k'], linewidths=[0.5])
```

```
129
130         return axis
```

C.2 入力データ

入力データは，(1) 粒子の分布，(2) 散布図のプロット設定の二つから構成される．

C.2.1 粒子分布

リスト 4 は，粒子の分布を指定するファイルである．

リスト 4: 散布図作成クラス (init_par.yml)

```
1  accelerator:
2      RF:                  200.0e6        # RF frequency [Hz]
3
4  particle:
5      e:          1.602176634E-19        # elelment charge [C]. minus for electron
6      m:           1.67262192E-27        # mass [kg]
7      N:                   10000         # macro paticles per period (or per bunh)
8
9  distribution:
10     Ib:                   2e-3         # Beam current [A]. minus for electron
11     shape:                  DC         # DC or Bunch
12     max_sigma:               6         # trancated Gaussion
13     N_bunch:                 5         # バンチの数
14     x:
15         av_posi:           0.0         # 平均 mu [m]
16         av_angle:          0.0         # 平均 x^\prime [rad]
17         alpha:            1.65         # Twiss parameter
18         beta:            0.051         #
19         epsilon: 2.894352E-05          # RMS emittance [m rad]
20
21     y:
22         av_posi:           0.0         # 平均 mu [m]
23         av_angle:          0.0         # 平均 x^\prime [rad]
24         alpha:            -0.8         # Twiss parameter
25         beta:            0.051         #
26         epsilon: 1.987654E-05          # RMS emittance [m rad]
27
28     z:
29         z0:                0.0         # 粒子の発生する座標z [m. すべてのマクロ粒子で同じ値. ]
30         E0:             35.0e+3        # 平均運動エネルギー [eV/u]
31         phi0:               180        # バンチ平均位相 [deg]
32         alpha:             0.0         # Twiss parameter
33         beta:             10.0         #
34         epsilon:         1.0e-6        # RMS emittance [rad eV]
```

C.2.2 散布図プロット設定

リスト 5 は，粒子の分布のプロットの設定ファイルである．

リスト 5: 散布図作成クラス (config.yml)

```
1  plot:
2      phase_space:
3          x:
4              position_range:   [-10.0, 10.0]      # プロット範囲 mm
5              angle_range:    [-200.0, 200.0]      #              mrad
6              marker:                  '.'         # 位相空間プロットの marker
7              size:                    4.0         # marker size
```

```
8                color_map:                  'jet'       # カラーマップ
9
10           y:
11               position_range:   [-10.0, 10.0]         # プロット範囲 mm
12               angle_range:     [-200.0, 200.0]        #              mrad
13               marker:                       '.'       # 位相空間プロットの marker
14               size:                        4.0        # marker size
15               color_map:                  'jet'       # カラーマップ
16
17           z:
18               E_unit:              {'unit':'keV/u', 'scale':1.0e-3}
19               dPhase_range:       [-5.0, 5.0]         # プロット範囲 deg
20               dE_range:           [-1.0, 1.0]         # 単位は              E_unit
21               marker:                       '.'       # 位相空間プロットの marker
22               size:                        4.0        # marker size
23               color_map:                  'jet'       # カラーマップ
```

C.3 実行バッチファイル

リスト 6 は，実行バッチファイルの例である．

リスト 6: 散布図作成クラス (run.ps1)

```
1 python program/mk_par_dist.py --seed 42 -i data/init_par.yml -o data/pars/init_par.pit
      --config data/config.yml
```

参考文献

[1] 山本昌志. 楕円.
http://www.yamamo10.jp/yamamoto/study/accelerator/theory/ellipse/index.php.

[2] 山本昌志. 正規分布.
http://www.yamamo10.jp/yamamoto/study/accelerator/theory/normal_dist/index.php.

[3] G. ストラング. 線形代数とその応用. 産業図書, 1992.
監訳者: 山口昌哉, 訳者: 井上昭.