LLL基底について

暗号理論のための格子の数学

１章の一部＆2章　の注意書き

信州大学大学院工学系研究科　岡崎裕之

〃　　水島大地（資料作成補助）

MENU

- 格子（1章）
- LLLアルゴリズム（2章）

格子　$L(\mathbf{B})$

- m次元ユークリッド空間の部分集合$L(\mathbf{B}) \subset \mathbf{R}^m$
- m次元ユークリッド空間の点の集合
- 基底 $\mathbf{B}$　n個の線形独立なベクトル
  の整数結合の集合
- 格子は基底の選び方によらない
  $L(\mathbf{B}) = L(\mathbf{B}')$　なる異なる基底が存在(だいたい)する
- 既知の基底から別の（都合のよい）基底を求めることが「基底変換（縮小）」の目的

格子　$L(\mathbf{B}) \subset \mathbf{R}^m$

$$\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbf{R}^{m \times n}$$

$$L(\mathbf{B}) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbf{Z}\}$$

言い換えると

$$\mathbf{x} = [x_1, \ldots, x_n] \in \mathbf{Z}^n$$

$$L(\mathbf{B}) = \{\mathbf{B}\mathbf{x}\}$$

用語の確認

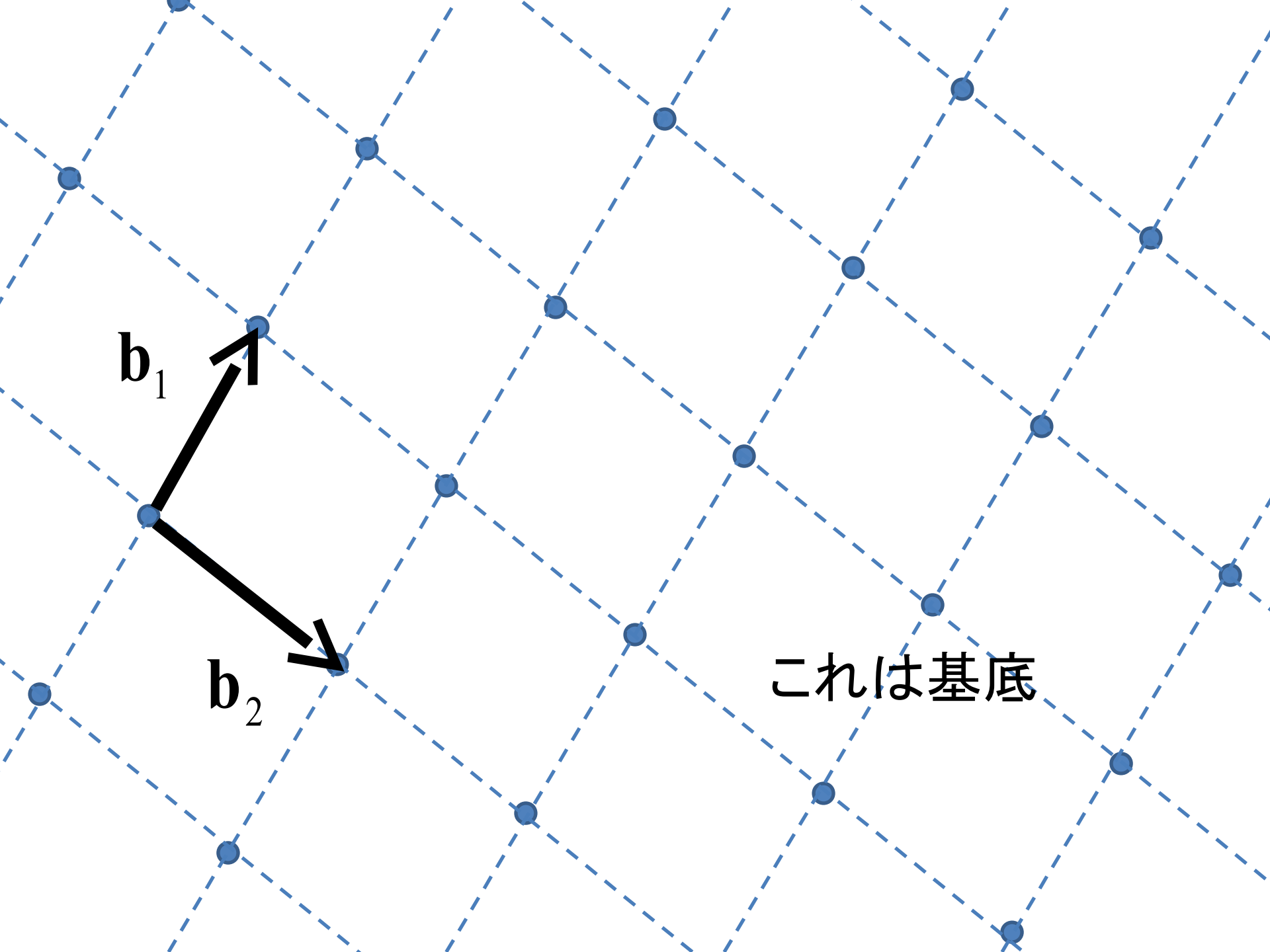
基底　$\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$

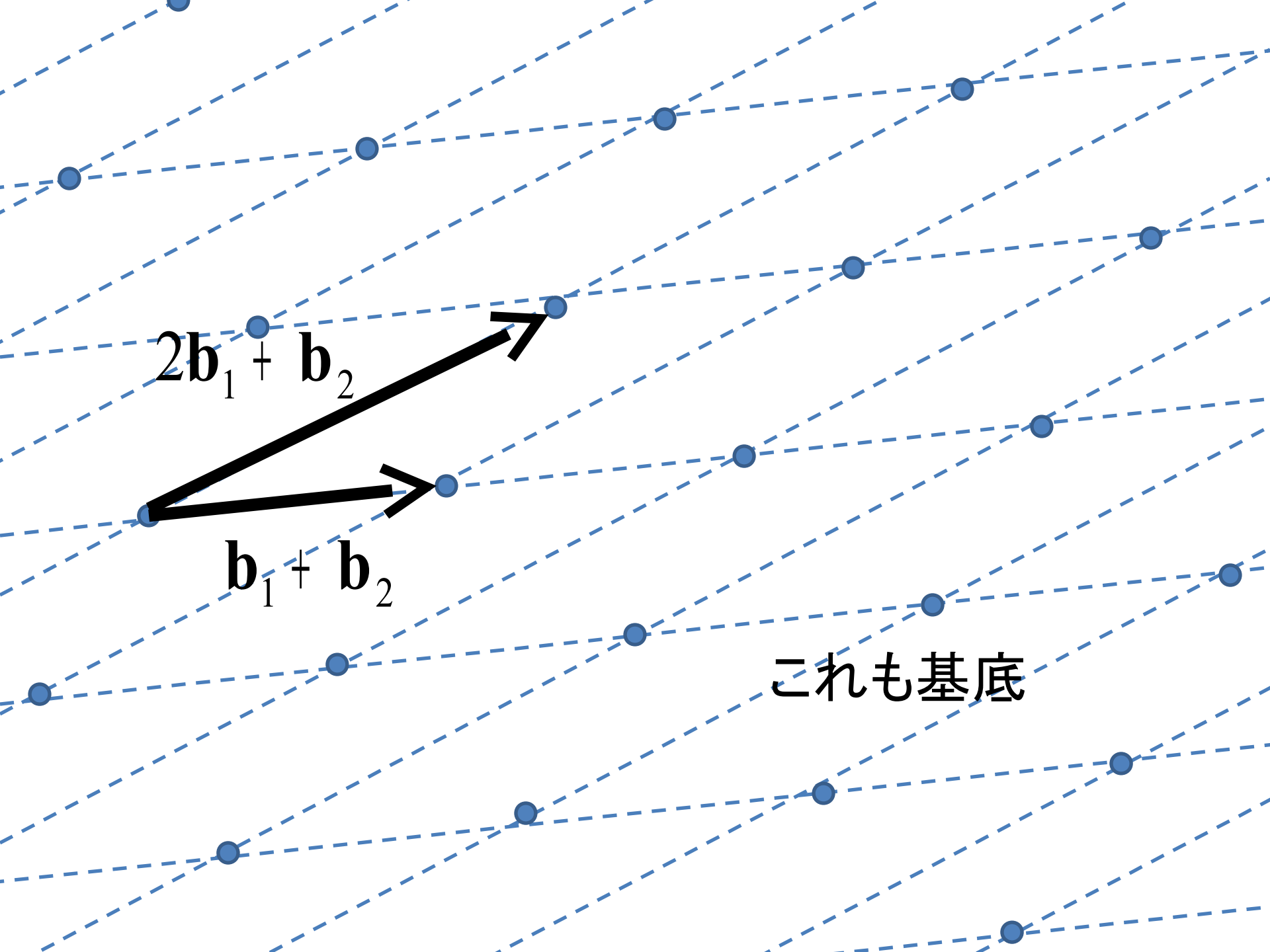
↑基底ベクトル

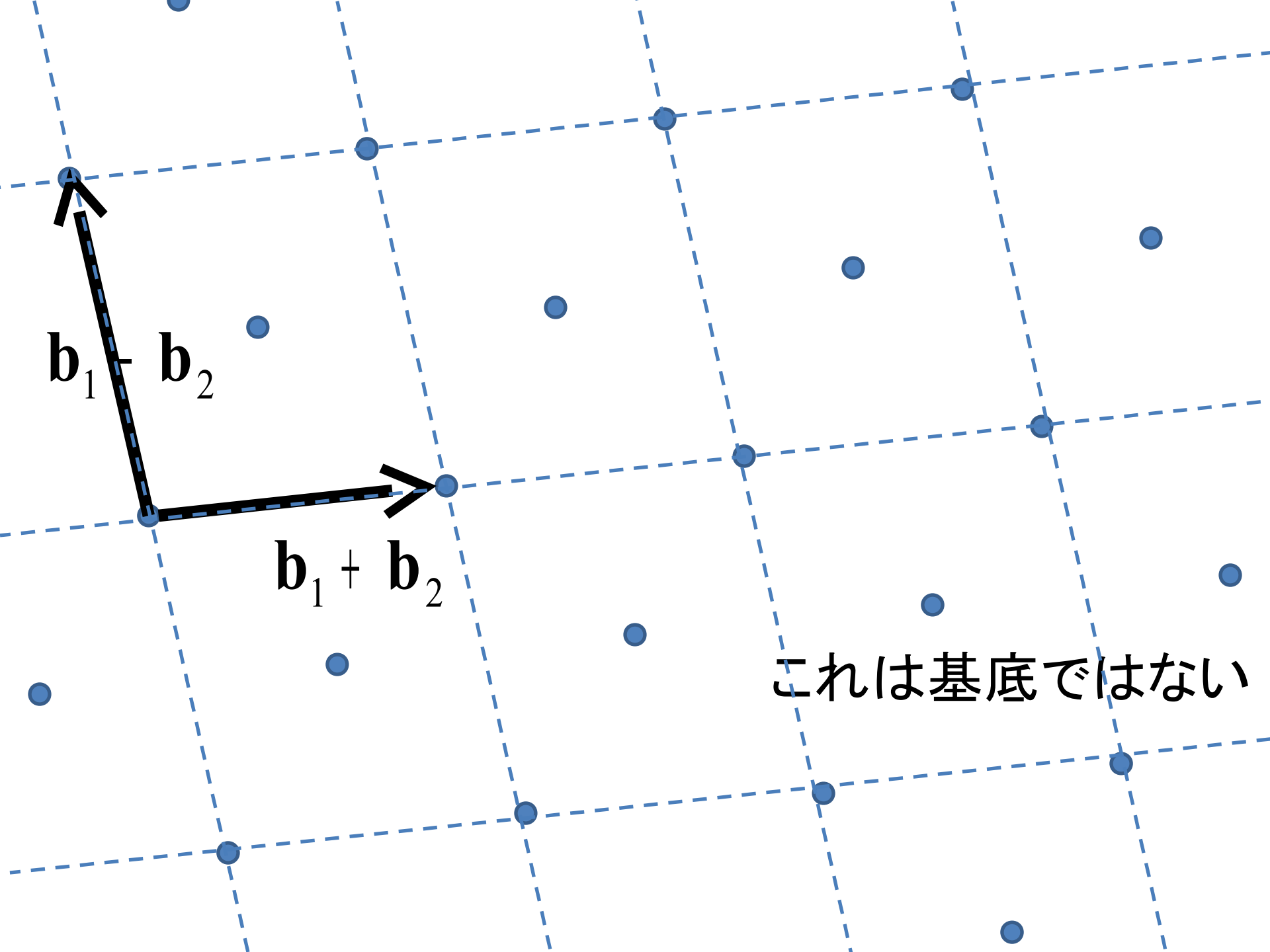
格子　$\boldsymbol{L}(\mathbf{B}) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbf{Z}\}$

格子ベクトルとか格子点とか　$\mathbf{y} \in \boldsymbol{L}(\mathbf{B})$

基底ベクトルは格子ベクトル！


$\mathbf{b}_1$
$\mathbf{b}_2$
これは基底


$2\mathbf{b}_1 + \mathbf{b}_2$
$\mathbf{b}_1 + \mathbf{b}_2$
これも基底


$\mathbf{b}_1 - \mathbf{b}_2$
$\mathbf{b}_1 + \mathbf{b}_2$
これは基底ではない

この例では

$$\Lambda = L([\mathbf{b}_1, \mathbf{b}_2]) = L([2\mathbf{b}_1 + \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2])$$

$[\mathbf{b}_1, \mathbf{b}_2], [2\mathbf{b}_1 + \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2]$ は Λ の基底

$$L([\mathbf{b}_1, \mathbf{b}_2]) \supset L([\mathbf{b}_1 - \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2])$$

$[\mathbf{b}_1 - \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2]$ は Λ の基底でなく

Λ の真部分格子の基底

$[\mathbf{b}_1, \mathbf{b}_2]$ や $[2\mathbf{b}_1 + \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2]$ と

$[\mathbf{b}_1 - \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2]$ は何が違うのか？

基底の特徴として
「基本領域（基本平行体）」
を考える

基底の基本平行体

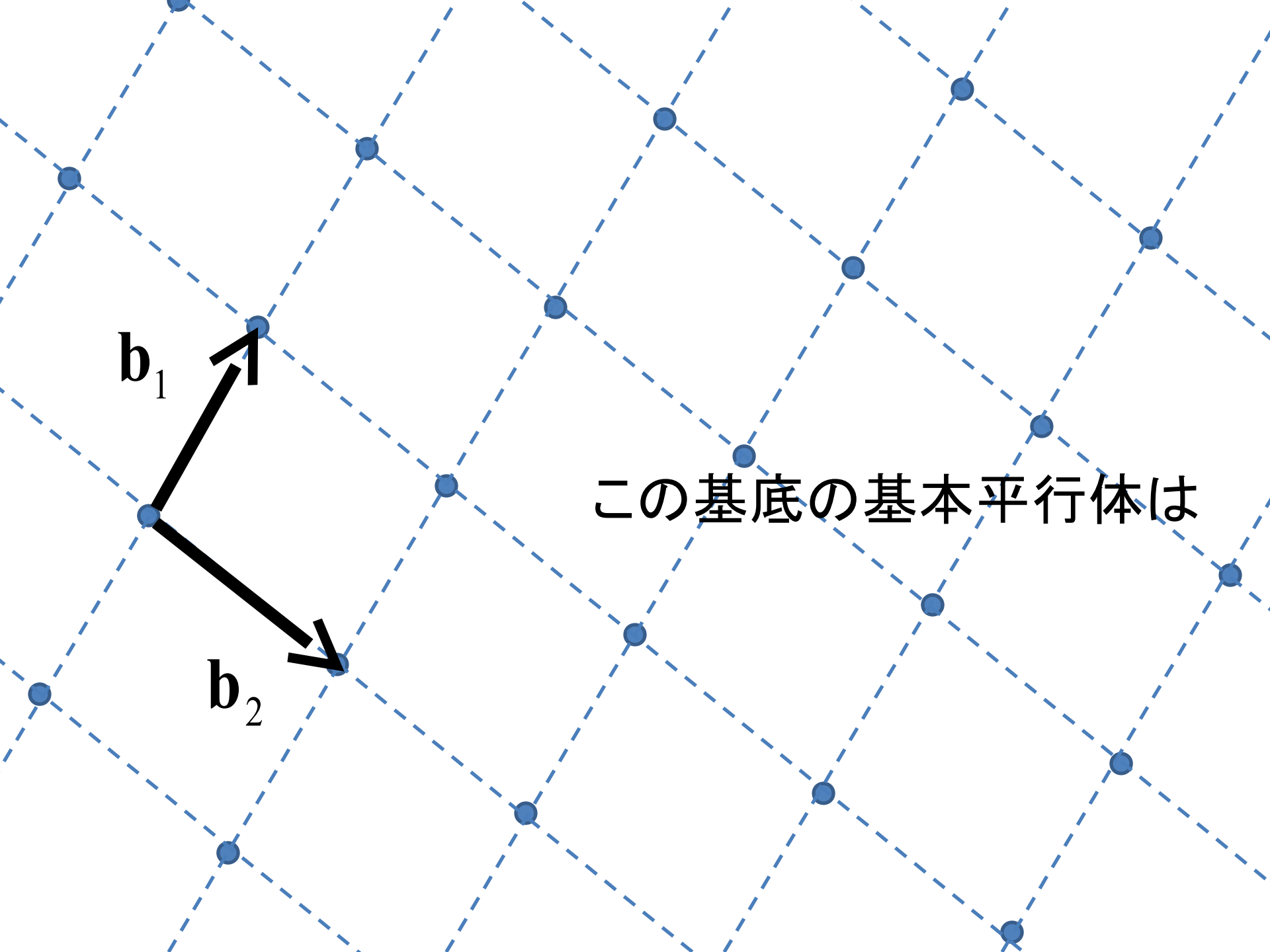
基底 $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$

基本平行体（半開平行体）

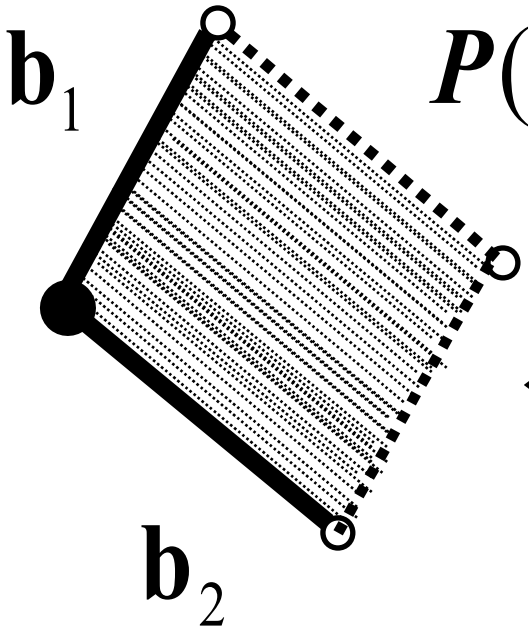
$$\boldsymbol{P}(\mathbf{B}) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i : 0 \leq x_i < 1\}$$

書いていませんが x_i は

おそらく実数



この基底の基本平行体は

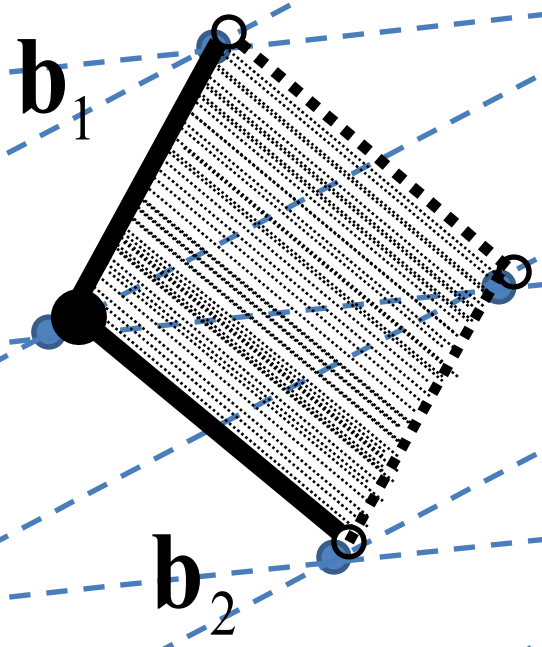


$P([\mathbf{b}_1, \mathbf{b}_2])$

←この領域

領域内に格子点は1個！

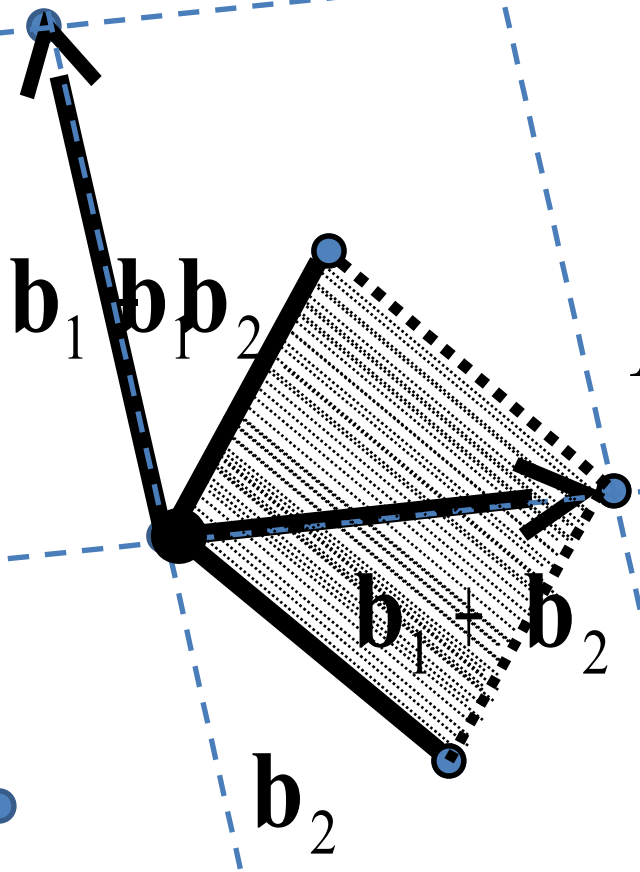
格子点の濃度は面積（体積）の逆数



$$\boldsymbol{P}([\mathbf{b}_1, \mathbf{b}_2])$$

面積は同じ

$$\boldsymbol{P}([2\mathbf{b}_1 + \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2])$$



$P([\mathbf{b}_1 - \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2])$ の面積は

$P([\mathbf{b}_1, \mathbf{b}_2])$ の2倍なので

$L([\mathbf{b}_1, \mathbf{b}_2])$ の格子点の半分は

$L([\mathbf{b}_1 - \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_2])$

に入っていない

基底の行列式

$\Lambda = L(\mathbf{B}) = L(\mathbf{B}')$ であれば

$P(\mathbf{B}), P(\mathbf{B}')$ の体積が等しいことを見た

じつは、基底 $\mathbf{B}$ の行列式 $det(\mathbf{B})$ は
$P(\mathbf{B})$ の体積になる

とうぜん $det(\mathbf{B}) = det(\mathbf{B}')$

Notation: $\Lambda = L(\mathbf{B})$ のとき $det(\Lambda) = det(\mathbf{B})$

行列式の計算

$\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ をグラムシュミット直行化法で与える

こういうのを

こう切り貼りすると簡単に面積求められる

でも行列式なんて普通にやれば計算できない？

ｸﾞﾗﾑｼｭﾐｯﾄ直行化法のﾒﾘｯﾄ

- 特定の値の行列式を持った行列を作れる
- ということは基底変換に使える
- ということで2章のアルゴリズムで使います
- 基底ベクトルの並び方と、選び方を変えてやれば特定の部分空間に対する射影とその他の部分に分解できる
- ということで使いやすい
- しかも効率的に計算できる

グラムシュミット直交化法

入力　$\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$

グラムシュミット直交化ベクトル　$\mathbf{B}^* = [\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*]$

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*$$

$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}$$

$$\det(\boldsymbol{L}(\mathbf{B})) = \prod_{i=1}^{n} \left\| \mathbf{b}_i^* \right\|$$

逐次最小

階数nの格子 Λ の逐次最小 $\lambda_1,\ldots,\lambda_i,\ldots,\lambda_n$

第 i 最小 $\lambda_i(\Lambda)$ とは
i 個の線形独立な格子ベクトルを含む
原点中心の最小の球の半径

格子 Λ の線形独立な格子ベクトルを短い順に並べると $v_1,\ldots,v_i,\ldots,v_n$ で長さがそれぞれ $\lambda_1,\ldots,\lambda_i,\ldots,\lambda_n$ というように取れるということ

$[v_1,\ldots,v_i,\ldots,v_n]$ は必ずしも Λ の基底ではない

ミンコフスキーの定理

$$\left(\prod_{i=1}^{n} \lambda_i \right)^{1/n} < \sqrt{n} \det(\Lambda)^{1/n}$$

とくに $\lambda_i < \sqrt{n} \det(\Lambda)^{1/n}$

Λ の最短非零格子ベクトルの長さが

$\sqrt{n} \det(\Lambda)^{1/n}$ でおさえられる

ミンコフスキーの定理

最短非零格子ベクトルの長さの上限の
“存在が示される”のであるが、最短ベクトル
そのものを与えてくれるわけでもないし、
実際のところもっと最短ベクトルの長さは
短いかもしれないので、計算したい人に
とってはあまりうれしくないので、
2章の近似アルゴリズムに続く！

SVP近似アルゴリズム（2章）

SVP(最短ベクトル問題とは)
基底 $\mathbf{B}$ が与えられたときに、
最短非零格子ベクトル、すなわち
長さ $\lambda_1(L(\mathbf{B}))$ のベクトル v_1 を
計算する問題

詳しくは面先生の資料で！

SVP近似アルゴリズム

- 2次元版確定的かつ効率的アルゴリズム
  （ガウスのアルゴリズム）
- 2次元版をn次元版に拡張する
- 拡張の副作用で「近似」アルゴリズムになる

（注）任意の効率的に計算可能なノルムに適用可能だが、ここではユークリッドノルムについて考える

2次元版SVPアルゴリズム

- ガウスのアルゴリズム
- 入力　　$[\mathbf{a},\mathbf{b}]$
- 出力　$[\mathbf{a}',\mathbf{b}']$

$$L([\mathbf{a},\mathbf{b}]) = L([\mathbf{a}',\mathbf{b}']) \,\&\, \lambda_1 = \|\mathbf{a}'\| \,\&\, \lambda_2 = \|\mathbf{b}'\|$$

- 多項式時間で計算可能
- やってることはベクトル版のユークリッド互除法、みたいな

簡約基底

- 2次元格子の簡約基底

  定義

  $[\mathbf{a},\mathbf{b}]$がノルム$\|\bullet\|$ について簡約されているとは

$$\|\mathbf{a}\|,\|\mathbf{b}\| \le \|\mathbf{a}+\mathbf{b}\|,\|\mathbf{a}-\mathbf{b}\|$$

  をみたすことである。

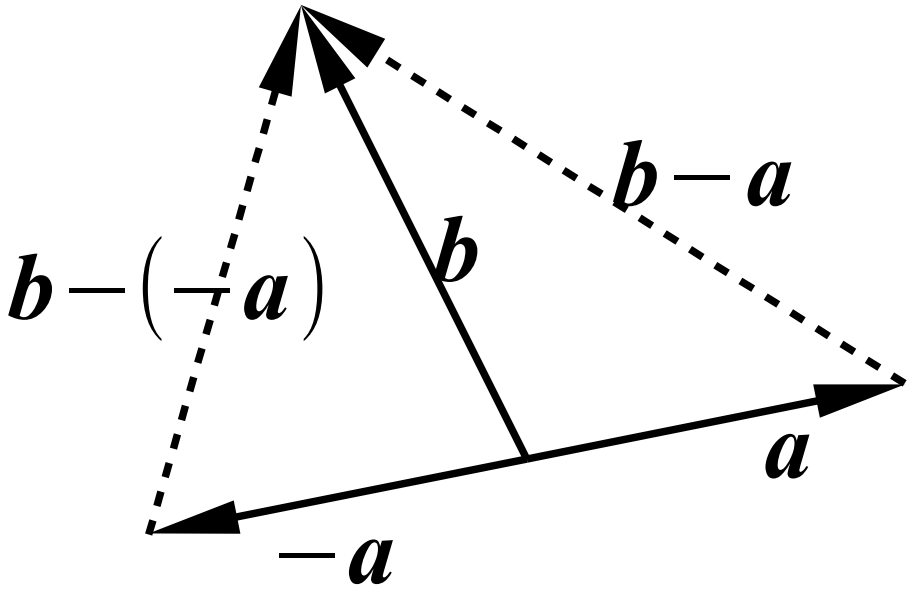
  となっているが実際はこうでしょう。

$$\|\mathbf{a}\|,\|\mathbf{b}\| \le \|\mathbf{b}-\mathbf{a}\|,\|\mathbf{b}-(-\mathbf{a})\|$$

簡約基底

- 2つのベクトルの差がどちらのベクトルよりもノルムが大きい

$$\|\boldsymbol{a}\|, \|\boldsymbol{b}\| \leq \|\boldsymbol{b}-\boldsymbol{a}\|, \|\boldsymbol{b}-(-\boldsymbol{a})\| = \|\boldsymbol{a}+\boldsymbol{b}\|$$



差が小さいノルム
持つなら基底として
入れ替えれば良い
（実際そういうこと
ガウスのアルゴリズム
でやってます）

簡約基底

- $[\mathbf{a},\mathbf{b}]$ が簡約　　iff

$$\lambda_1 = \|\mathbf{a}\| \,\&\, \lambda_2 = \|\mathbf{b}\|$$

ただし $\|\mathbf{a}\| < \|\mathbf{b}\|$ を仮定

証明は教科書追ってください

ガウスのアルゴリズム

- 初期化：入力された基底が簡約なら即終了
  簡約でなければ
  「整列」させてLOOPへ
- LOOP:　基底を「割り算」っぽいことしてノルムの小さい基底ベクトルに差し替える
  基底を整列させる
  整列させた基底が簡約なら即終了
  簡約でなければLOOPへ

整列とは

- 定義

  基底$[\mathbf{a},\mathbf{b}]$が整列されているとは

$$\|\mathbf{a}\| \leq \|\mathbf{a}-\mathbf{b}\| \leq \|\mathbf{b}\|$$

  をみたすことである。

初期化のステップでは入力された基底をうまく細工して整列された基底にしてLOOPへ行く。

これは簡単なので省略

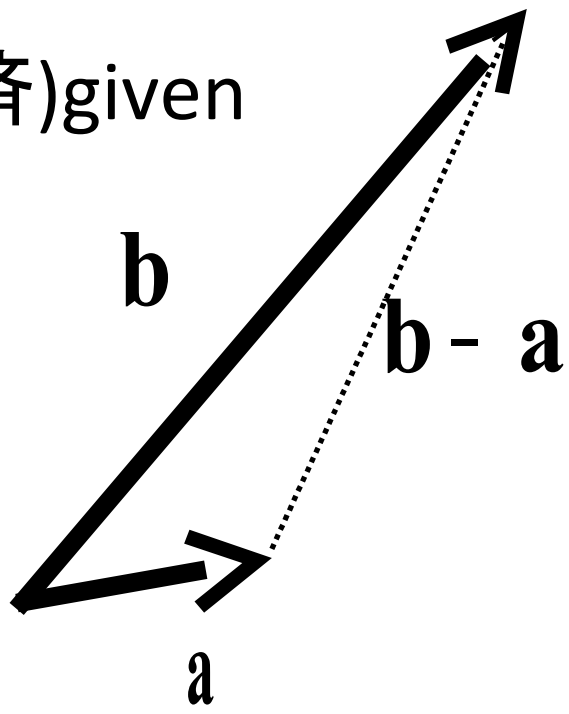
整列

- $\|\mathbf{b}\| \leq \|\mathbf{a}\|$ ならば $\mathbf{a}, \mathbf{b}$ をスワップ
- $\|\mathbf{b} - \mathbf{a}\| \leq \|\mathbf{b} + \mathbf{a}\|$ ならば符号反転 $\mathbf{b} := -\mathbf{b}$
- これで基底は簡約、もしくは整列される

LOOPのステップ

- LOOPの最初では必ず基底は整列されている
- 基底を「割り算」っぽいことしてノルムの小さい基底ベクトルに差し替える

$[\mathbf{a}, \mathbf{b}]$ (整列済)given



LOOPのステップ

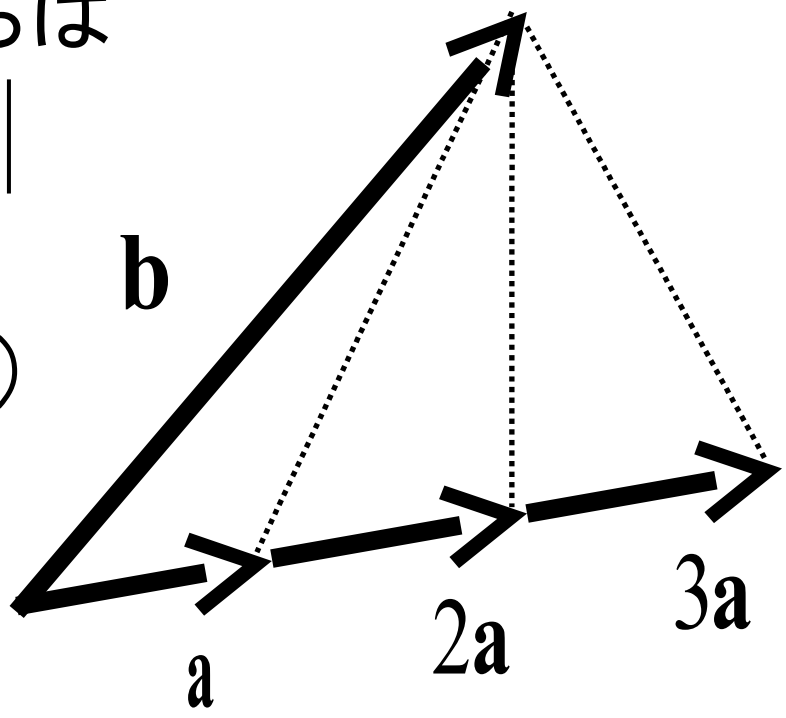
$\|\mathbf{b} - \mu\,\mathbf{a}\|$ が最小になる μ を探す（割り算っぽい）

μ は2分探索で探すことができる

$\|\mathbf{x}\| \le \|\mathbf{x} + \mathbf{y}\|$ ならば

$\|\mathbf{x} + \mathbf{y}\| \le \|\mathbf{x} + \alpha\,\mathbf{y}\|$

$(1 \le \alpha)$ なので　（補題２．１）



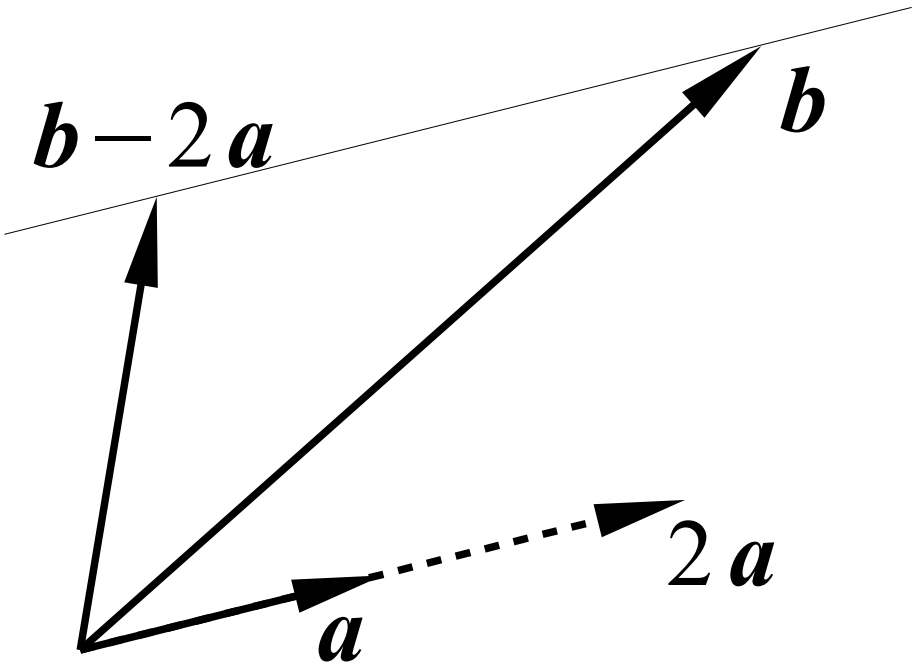
この場合は微妙ですが

$\mu = 2$

LOOPのステップ

基底ベクトルの入れ替え

$\boldsymbol{b}$ を $\boldsymbol{b}-\mu\boldsymbol{a}$ で置き換える



$\boldsymbol{b}-\mu\boldsymbol{a}$を

$\boldsymbol{b}$ の $\boldsymbol{a}$

による剰余と

みればこれは

互除法でしょ

基底が簡約なら終了

そうでなければ整列

してLOOP続行

（この例ではこれで

終わり）

LOOPのステップ

- LOOP最初で基底は整列されている
- 割り算によって得られた剰余のベクトルは必ずノルムが小さくなる
  （さもなければ基底は簡約されているはず）
- ということで有限回（実際のところ多項式オーダー）のLOOPで停止する。

n 次元へ拡張(LLL アルゴリズム)

- n 次元の多項式時間アルゴリズムが欲しい！

→普通にやると実行時間が抑えられない

（ガウスのアルゴリズムでいう

"整列”がうまくいかない）

- パラメータ δ を用いて簡約基底を”ゆるく”定義

→ δLLL 簡約基底

簡約基底(2 次元版)

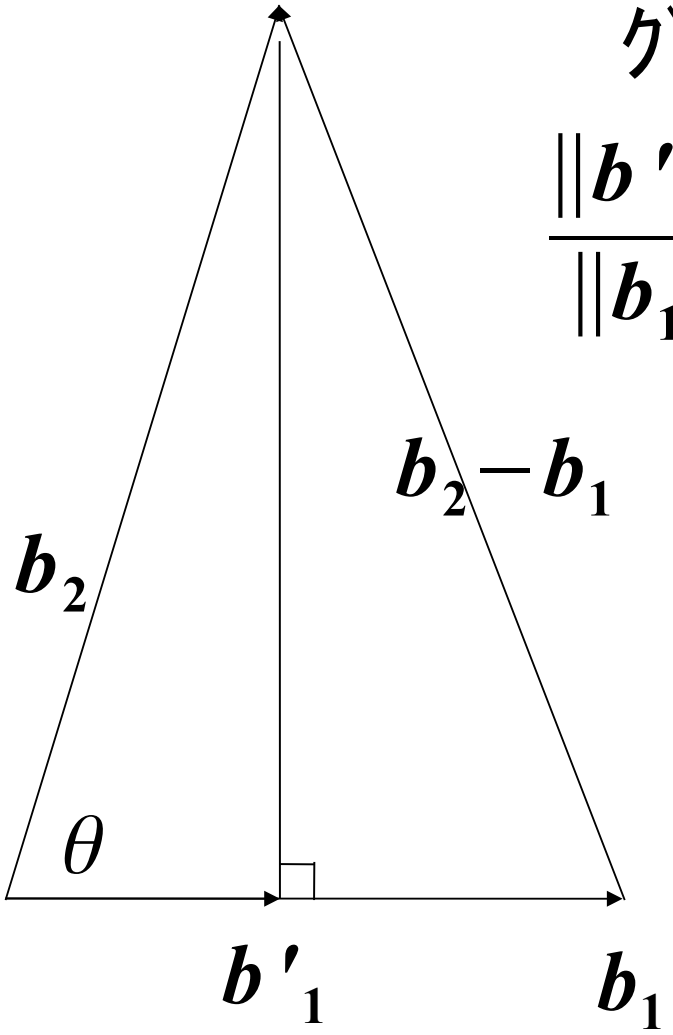
- 2 次元格子基底の簡約条件(おさらい)

$$\|\boldsymbol{b}_1\|, \|\boldsymbol{b}_2\| \leq \|\boldsymbol{b}_1 + \boldsymbol{b}_2\|, \|\boldsymbol{b}_1 - \boldsymbol{b}_2\|$$

- 再定義(別の表現)すると

$$\mu_{2,1} = \frac{\langle \boldsymbol{b}_2, \boldsymbol{b}_1 \rangle}{\langle \boldsymbol{b}_1, \boldsymbol{b}_1 \rangle} \leq \frac{1}{2} \quad \wedge \quad \|\boldsymbol{b}_1\| \leq \|\boldsymbol{b}_2\|$$

幾何的に…



グラムシュミット係数 ＝ 射影成分

$$\frac{\|b'_1\|}{\|b_1\|} = \frac{\|b_2\|}{\|b_1\|}\cos\theta = \frac{\langle b_2, b_1\rangle}{\langle b_1, b_1\rangle}$$

これが 1/2 以下だったら
当然”差”の方が長いか同じ
↓
簡約されている

δLLL 簡約基底

1. ｸﾞﾗﾑｼｭﾐｯﾄ係数について

$$|\mu_{i,j}| \leq \frac{1}{2} \quad (i > j)$$

2. 基底のうち、任意の連続したベクトル対について

$$\delta \|\pi_i(\boldsymbol{b}_i)\|^2 \leq \|\pi_i(\boldsymbol{b}_{i+1})\|^2 \quad \left(\frac{1}{4} < \delta < 1\right)$$

π_i : $span(\boldsymbol{b}_i^*, \boldsymbol{b}_{i+1}^*, \ldots, \boldsymbol{b}_n^*)$ への射影演算

LLL アルゴリズム

- 大まかに2つの部分に分けられる

a. 簡約ステップ（条件 1. をみたすため）

各 i, j (i > j) で

$$\boldsymbol{b}_i := \boldsymbol{b}_i - c\,\boldsymbol{b}_j$$

b. 交換ステップ（条件 2. をみたすため）

ある i が条件 2. をみたさなければ

$$swap\left(\boldsymbol{b}_i, \boldsymbol{b}_{i+1}\right)$$

して a. にもどる

LLL アルゴリズム

- LLL アルゴリズムは δLLL 簡約基底を出力する

→最短ベクトルは求められない

→多項式時間で終了 & 解の上界が分かっている

実行例(3 次元空間)

- 入力、δ は

$$\boldsymbol{b}_1=(4,1,2),\quad \boldsymbol{b}_2=(4,7,2),\quad \boldsymbol{b}_3=(3,1,7)$$

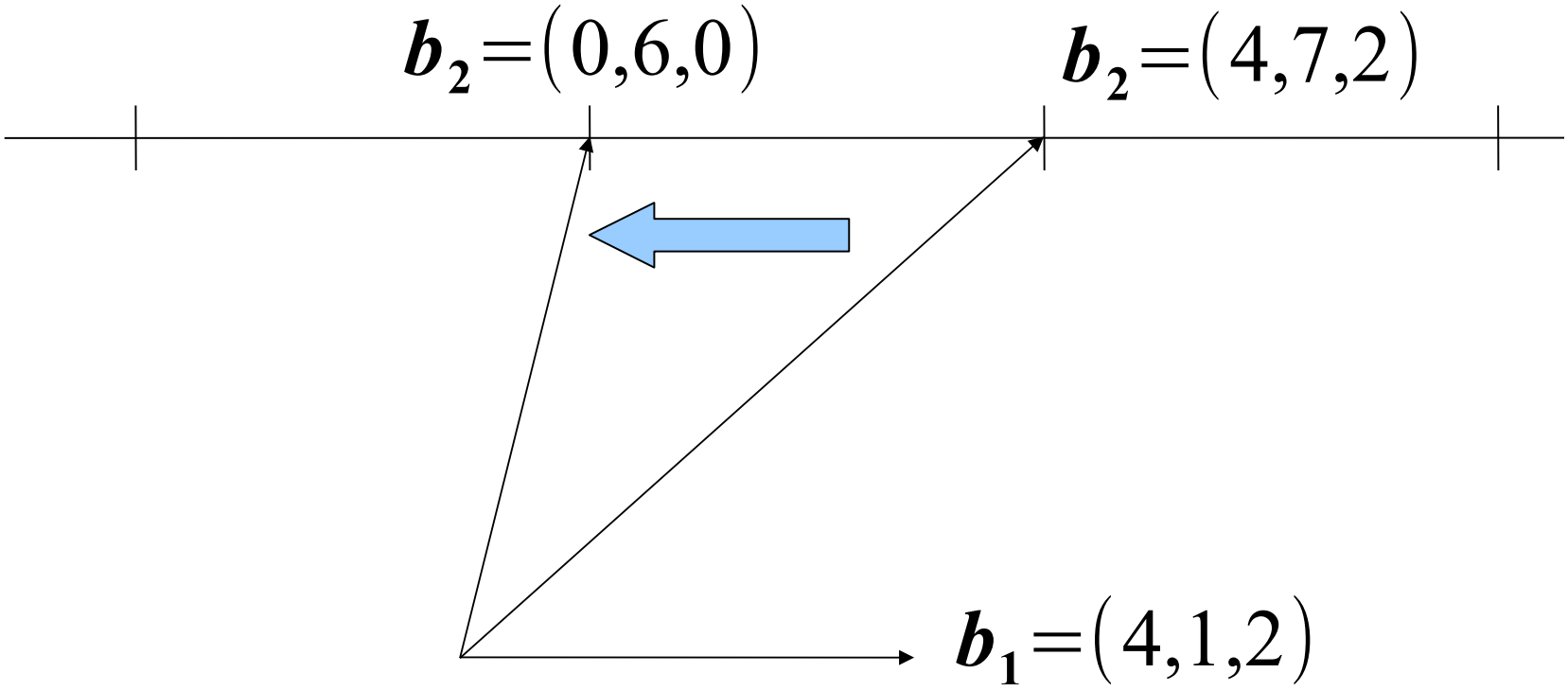
$$\delta=\left(\frac{1}{4}\right)+\left(\frac{3}{4}\right)^{\frac{3}{2}}\approx 0.9$$

- グラムシュミット直交化すると

$$\boldsymbol{b}_1^*=(4,1,2),\quad \boldsymbol{b}_2^*=\left(-\frac{8}{7},\frac{40}{7},-\frac{4}{7}\right),\quad \boldsymbol{b}_3^*=\left(-\frac{11}{5},0,\frac{22}{5}\right)$$

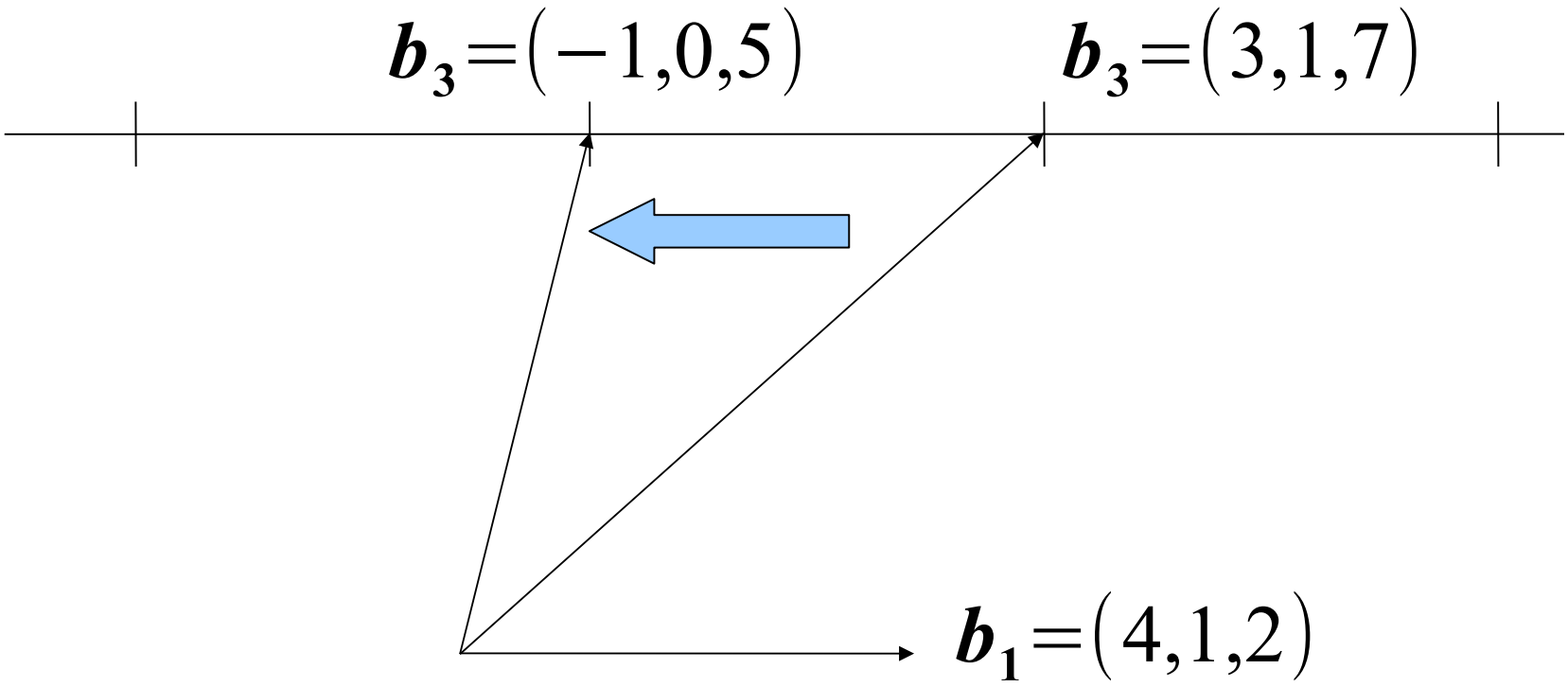
簡約ステップ（b_2 ）

$$b_2 := b_2 - b_1 = (0,6,0)$$



簡約ステップ（$\boldsymbol{b}_3$ ）

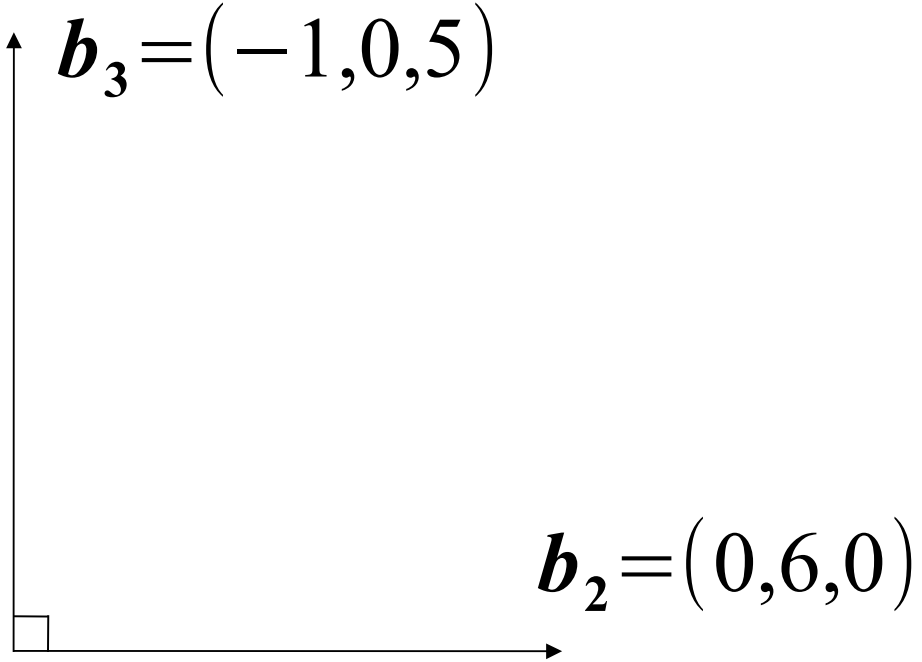
$$\boldsymbol{b}_3 := \boldsymbol{b}_3 - \boldsymbol{b}_1 = (-1,0,5)$$



簡約ステップ（$\boldsymbol{b}_3$ ）

直交しているので操作の必要がない

$$\boldsymbol{b}_3 := \boldsymbol{b}_3 - (0 * \boldsymbol{b}_2) = (-1,0,5)$$



交換ステップ（$\boldsymbol{b}_1$ と$\boldsymbol{b}_2$ ）

$$\boldsymbol{b}_1=(4,1,2),\quad \boldsymbol{b}_2=(0,6,0)$$

より

$$\delta\|\pi_1(\boldsymbol{b}_1)\|^2=\delta\|(4,1,2)\|^2\approx 18.9$$

$$\|\pi_1(\boldsymbol{b}_2)\|^2=\|(0,6,0)\|^2=36$$

だから、交換しなくていい

交換ステップ($\boldsymbol{b}_2$ と$\boldsymbol{b}_3$)

$$\boldsymbol{b}_2=(0,6,0),\quad \boldsymbol{b}_3=(-1,0,5)$$

より

$$\delta\|\pi_2(\boldsymbol{b}_2)\|^2=\delta\|(-\frac{8}{7},\frac{40}{7},-\frac{4}{7})\|^2\approx 30.9$$

$$\|\pi_2(\boldsymbol{b}_3)\|^2=\|(-\frac{15}{7},-\frac{2}{7},\frac{31}{7})\|^2\approx 24.3$$

だから、交換して簡約ステップに戻る

簡約ステップ（今回は何もしない）

$$\boldsymbol{b_1}=(4,1,2),\quad \boldsymbol{b_2}=(-1,0,5),\quad \boldsymbol{b_3}=(0,6,0)$$

ここで、グラムシュミット係数をみると

$$|\mu_{2,1}|=|\frac{2}{7}|<\frac{1}{2},\quad |\mu_{3,1}|=|\frac{2}{7}|<\frac{1}{2},\quad |\mu_{3,2}|=|-\frac{1}{20}|<\frac{1}{2}$$

なので操作の必要がない

よってこれを δLLL 簡約基底として出力し終了

簡約ステップは正しい？

- 簡約ステップの各操作は基底のうち2つしかみない

→それ以外の部分のｸﾞﾗﾑｼｭﾐｯﾄ係数は大丈夫？

- 直交化基底は簡約ステップの前後で不変

（というかｸﾞﾗﾑｼｭﾐｯﾄ直交化もどきをやってるだけ）

なので…大丈夫！

実行時間評価

- ループ回数 = 交換ステップの回数
- 実際、交換ステップの条件を（δ で）甘くしてる
- δ < 1 ならば多項式で抑えられる

→証明は…教科書参照

SVP をどのくらい近似できてる？

- δLLL 簡約基底であるならば

$$\|\boldsymbol{b}_1\| \leq \left(2/\sqrt{4\delta - 1}\right)^{n-1} \lambda_1$$

- とくに　$\delta = \left(1/4\right) + \left(3/4\right)^{\frac{n}{(n-1)}}$　ならば

$$\|\boldsymbol{b}_1\| \leq \left(2/\sqrt{3}\right)^{n} \lambda_1$$

まとめ

- 原理はガウスのアルゴリズムと同じ
- あくまで近似解を出力
- それでも、多項式時間で SVP の近似解を与える

付録：PARI-GP での実行結果

```
? b = [4,4,3;1,7,1;2,2,7]
%1 =
[4 4 3]
[1 7 1]
[2 2 7]

? b * qflll(b)
%2 =
[4 -1 0]
[1 0 6]
[2 5 0]
```