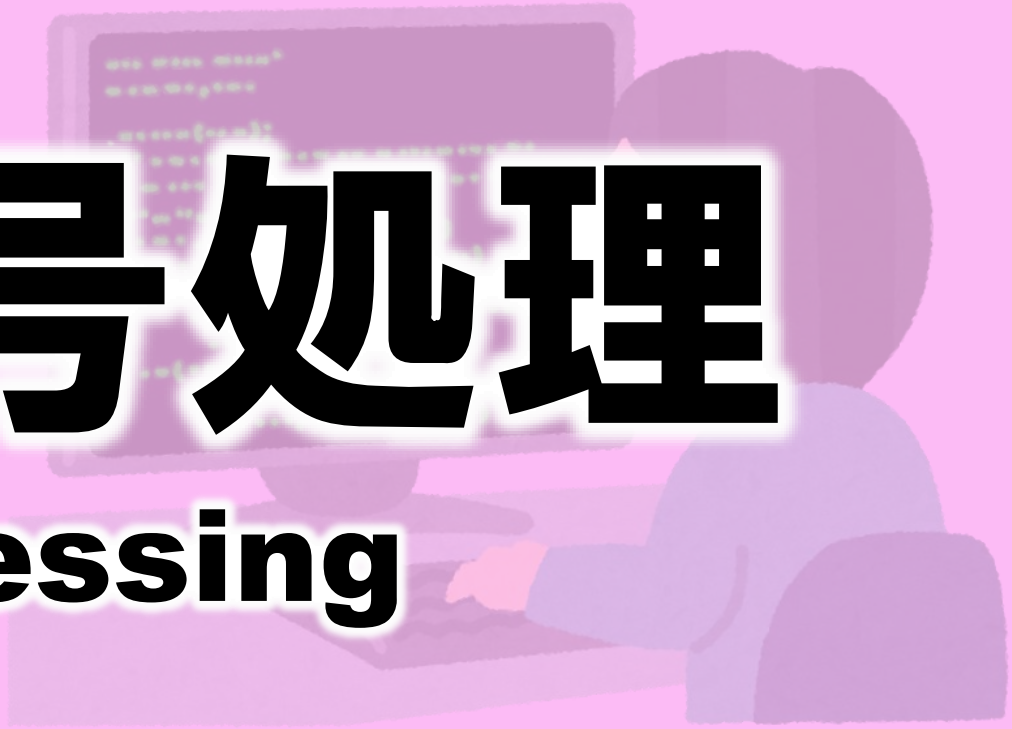
2017年6月24日更新



7. 信号処理

Signal processing

東北大学 大学院工学研究科
嶋田 慶太

TOHOKU UNIVERSITY

今日の内容はほとんどフーリエ変換

フーリエ変換

記法の一例

$$\mathcal{F}[f_g(t)](f) = F_g(f) = \int_{-\infty}^{\infty} f_g(t) \exp(-\mathrm{i}\, 2\pi f t)\, \mathrm{d}t$$

- $\mathcal{F}$: スクリプト体のFで Fourier transformを表す.
- f_g: 被変換関数
- f: 変換後の変数（省略可）
- $f_g(t)\exp(-\mathrm{i}\, 2\pi f t)$: 関数 f_g と, $\cos(2\pi f t)$ と $\mathrm{i}\sin(2\pi f t)$ との **内積** を取る（実部と虚部は分けて計算してよい）

▼

被変換関数 f_g に含まれる, $\cos(2\pi f t)$ と $\mathrm{i}\sin(2\pi f t)$ の **成分の強度** が得られる

離散フーリエ変換

$$\mathbf{v} \cdot \mathbf{u}_i$$

- $\mathbf{v}$: 被変換ベクトル
- $\mathbf{u}_i$: 関数の場合の cosやsin に対応する基底ベクトル（正規直交）

ベクトルの内積

内積とは

- 二つのベクトル・関数の**関係性**を表す指標

  さらに「長さ」で除すと，関係性が規格化される．

2次元では…

$$\mathbf{v}_1 = (x_1, y_1)$$

$$\mathbf{v}_2 = (x_2, y_2)$$

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = x_1 x_2 + y_1 y_2$$

- 0 なら互いに独立（直交）すなわち **関係しないといえる**

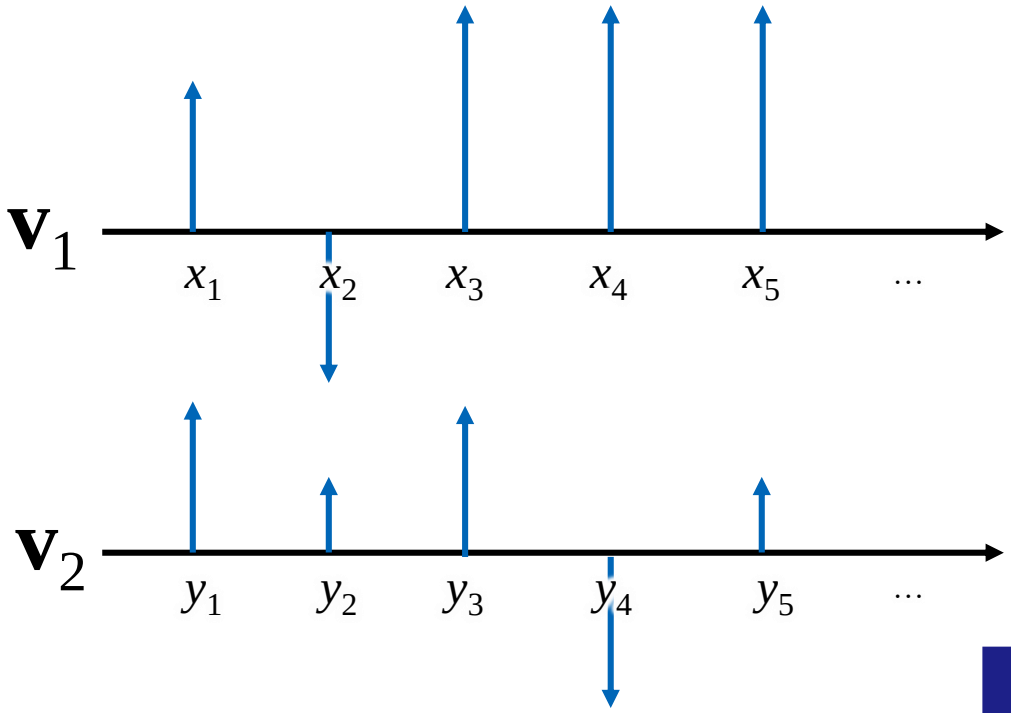
$$\mathbf{v}_1 \cdot \mathbf{v}_2 = 0 \leftrightarrow \mathbf{v}_1 \perp \mathbf{v}_2$$

しかし，2次元ベクトルはなまじ見えるため，
多次元を想像しずらい

多次元ベクトルについては

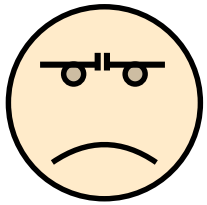
高次元ベクトルの表現

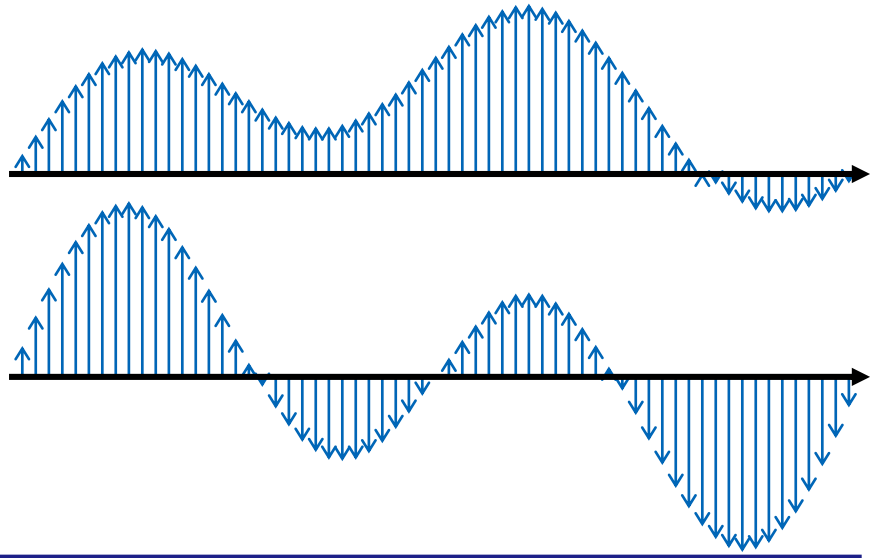
n (>3) 次元以上の表現



$$(\mathbf{v}_1,\mathbf{v}_2) = \sum_{i=1}^{n} x_i y_i$$

5次元以上？





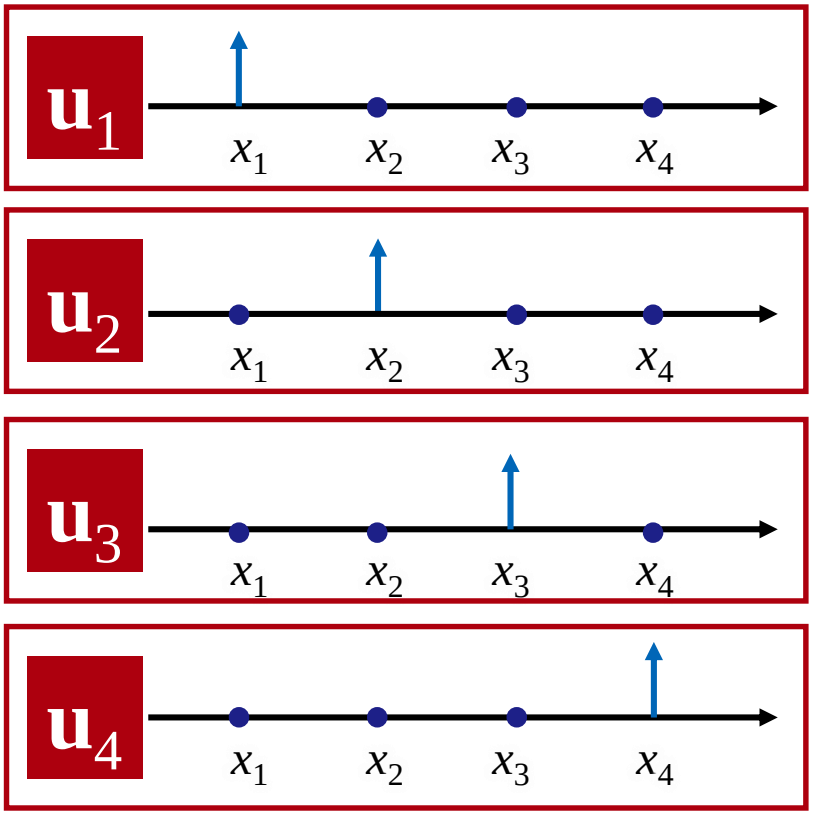
関数の内積 $\int f_1(x) f_2(x)\,\mathrm{d}x$

とのイメージ的な結びつきが得やすい

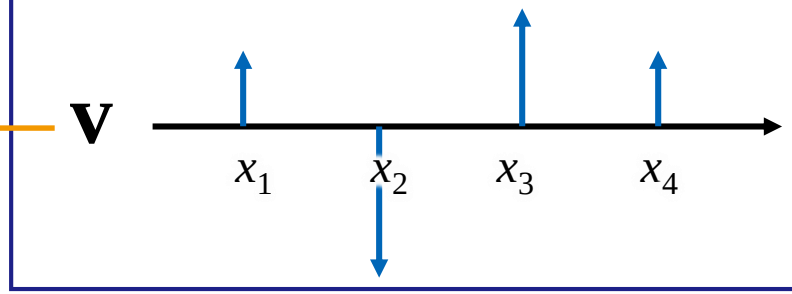
2次元・3次元だと
どうしても幾何学的なイメージに縛られがち

基底（例えば4次元の場合）

基底 N次元空間を線形結合で表現することのできるベクトルの組



例えばベクトル$\mathbf{v}$について



内積 $\mathbf{v}\cdot\mathbf{u}_i$ $\mathbf{v}$が$\mathbf{u}_i$の成分をどれだけ持つか

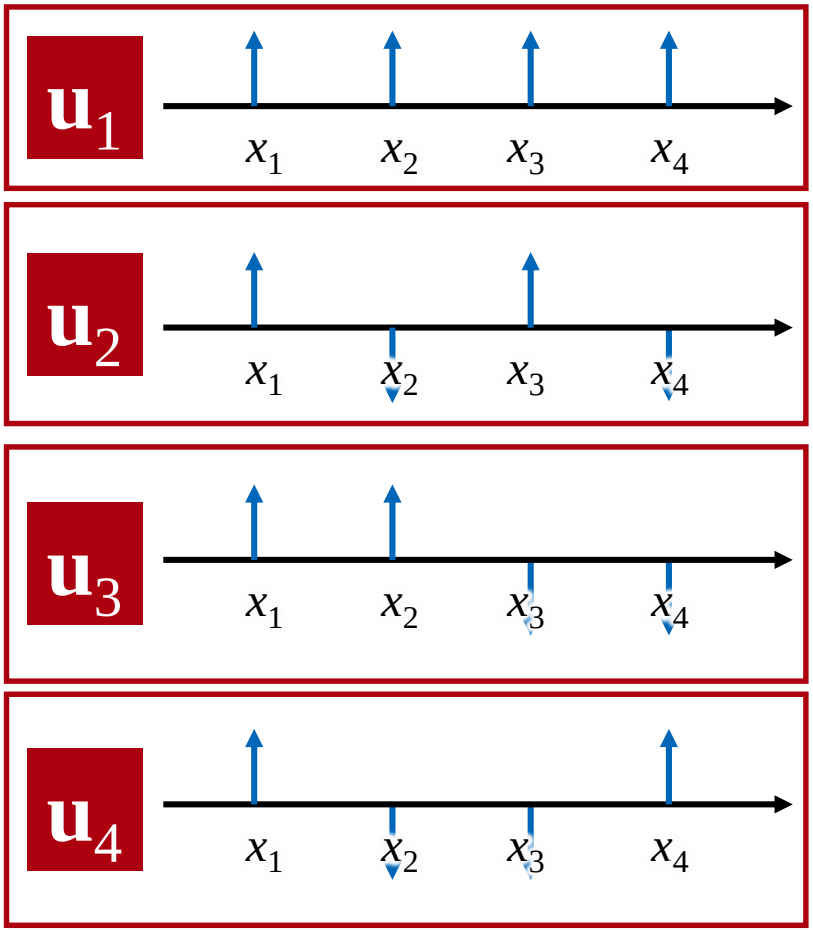
正規直交基底なら

$$\mathbf{v} = \sum_{i=1}^{n} (\mathbf{v}\cdot\mathbf{u}_i)\mathbf{u}_i$$

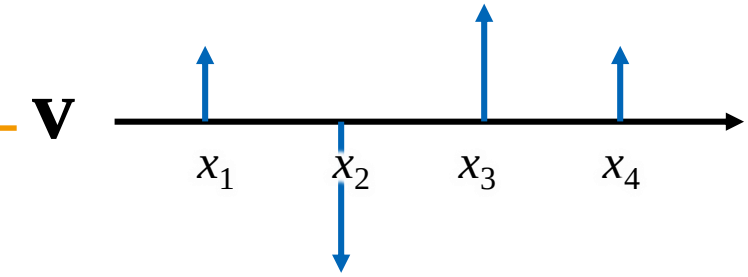
内積で成分に分解◂内積を取るまでもない？

基底の取り方

正規直交基底をこんな感じで選べる



例えばベクトルvについて



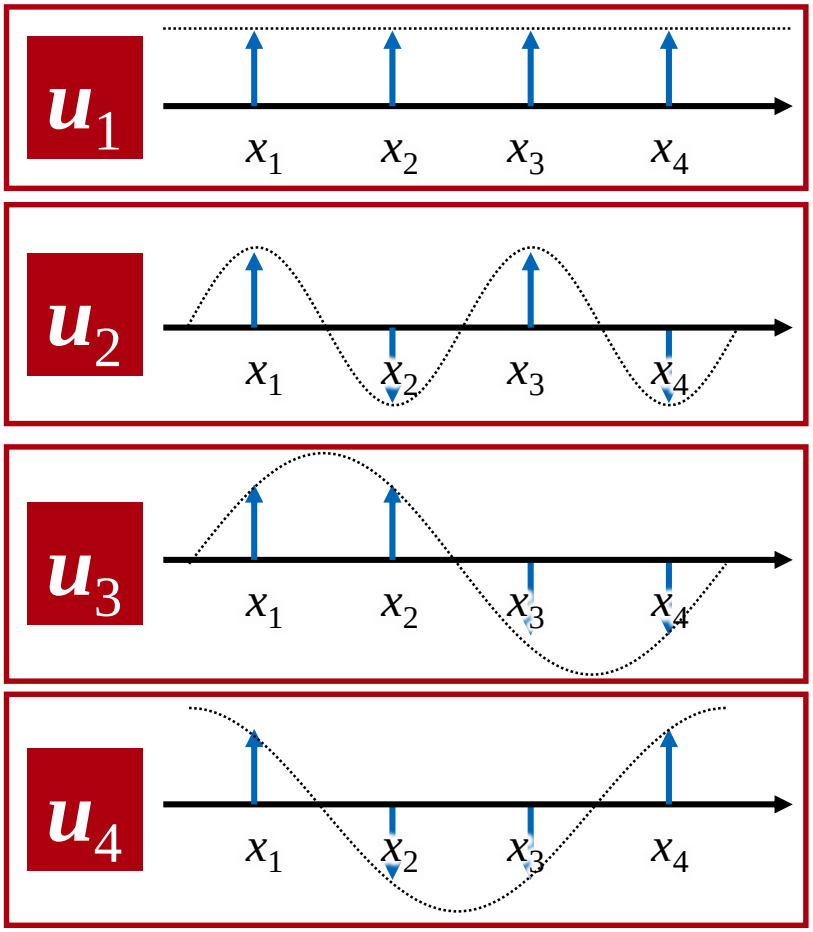
内積 $\boldsymbol{v}\cdot\boldsymbol{u}_i$ vがu_iの成分をどれだけ持つか

$$\mathbf{v} = \sum_{i=1}^{n} (\mathbf{v}\cdot\mathbf{u}_i)\mathbf{u}_i$$

内積で成分に分解 ◂ このu_iになんの意味が？

先ほどの基底の意味

直交基底をこんな感じで選べる

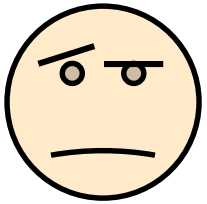


内積 $\mathbf{v}\cdot\mathbf{u}_i$ $\mathbf{v}$が$\mathbf{u}_i$の成分を
どれだけ持つか

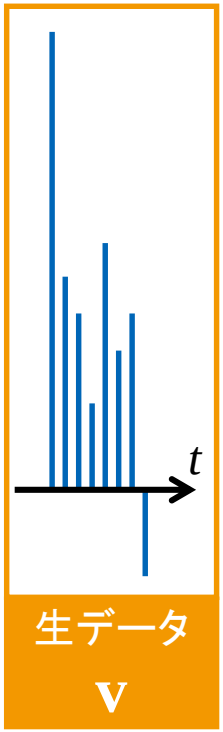
三角関数に対応付け，
分解できる

$$\mathbf{v} = \sum_{i=1}^{n} (\mathbf{v}\cdot\mathbf{u}_i)\mathbf{u}_i$$

離散フーリエ変換



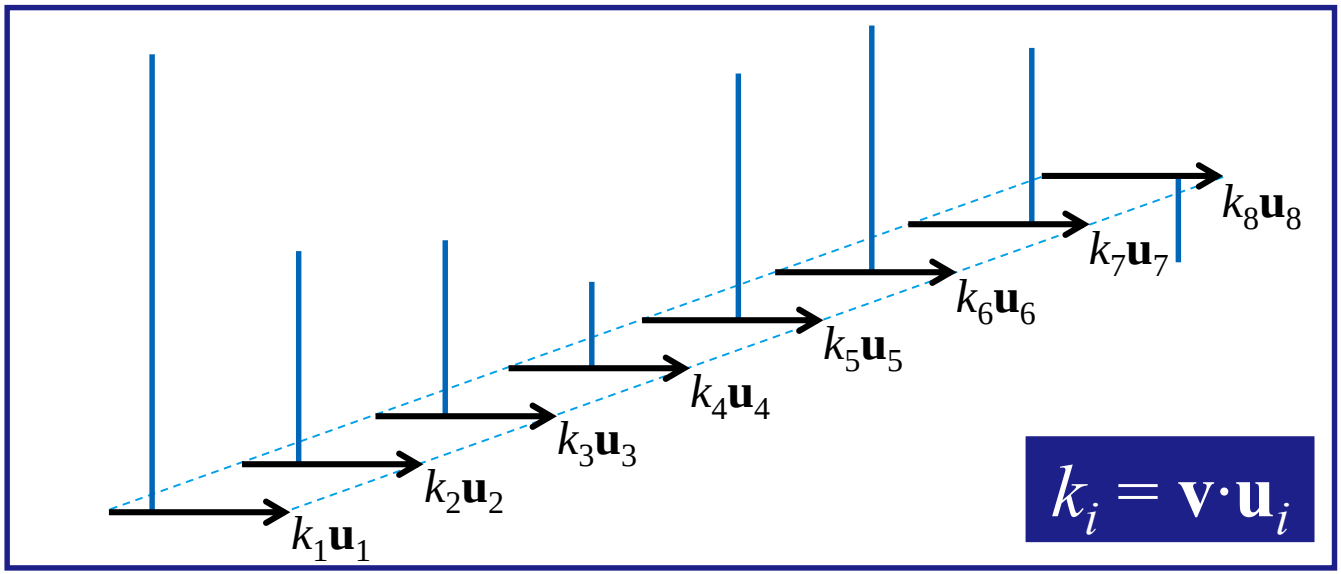
分解のイメージ

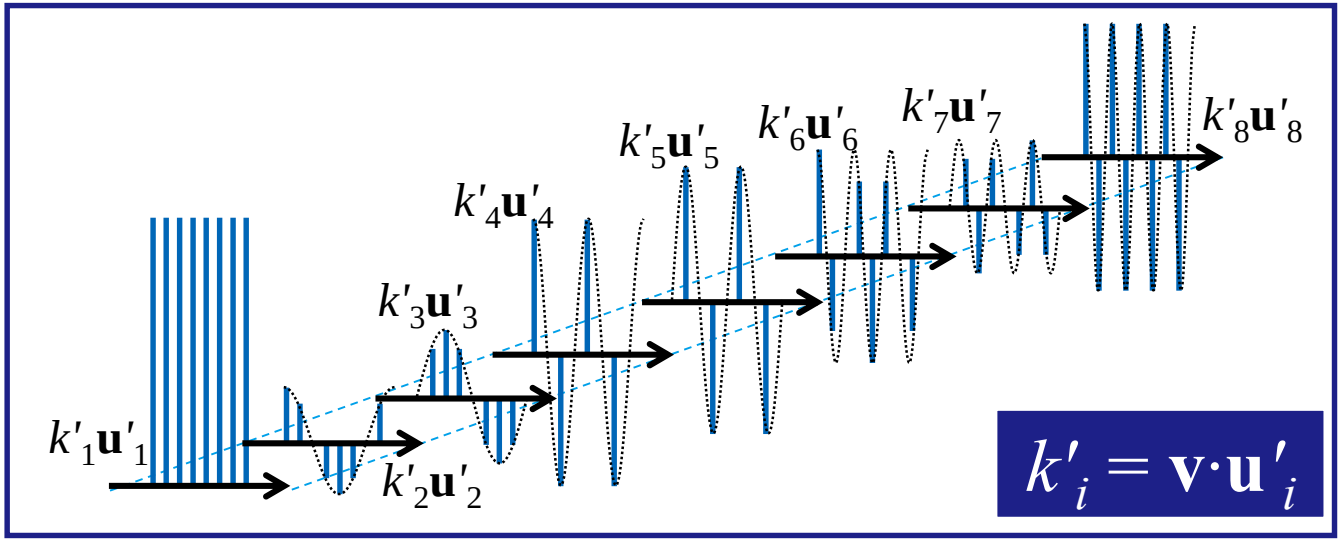


時間ごとの
単位ベクトル$\mathbf{u}_i$
とその強度k_i
（自明な分解）

内積
により分解

周波数ごとの
単位ベクトル$\mathbf{u}'_i$
とその強度k'_i





離散フーリエ変換のための基底の選択

基底ベクトルの選択

$$\mathbf{u}_f = \left\{ \begin{array}{c} \exp\left(-\mathrm{i}\frac{2\pi}{N} f \times 0\right) \\ \exp\left(-\mathrm{i}\frac{2\pi}{N} f \times 1\right) \\ \vdots \\ \exp\left(-\mathrm{i}\frac{2\pi}{N} f (N-1)\right) \end{array} \right\}$$

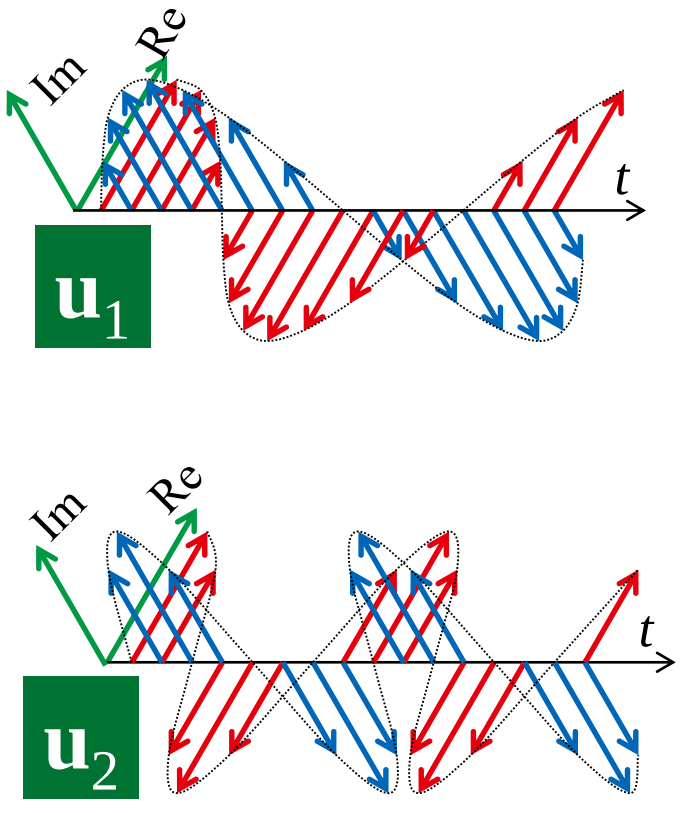
iは虚数単位

$f = 0, 1, \ldots, N\text{-}1$

データ点数

列ベクトルに単位円の点を対応させていく

$N = 16$ の一部



基底の直交性

複素数の内積なので，片方は複素共役であることに注意

$$\mathbf{u}_f \cdot \mathbf{u}_\varphi = \sum_{t=0}^{N-1} \exp\left(-\mathrm{i}\frac{2\pi}{N}(f-\varphi)t\right)$$

$$= N\delta_{f\varphi}$$

クロネッカーのデルタ

いずれの基底 $\boldsymbol{u}_f$ も独立

単位円の正N角形を考えて和を取る
fとφが異なれば，正多角形，星形多角形を取るので総和は0

$\mathbf{v} \bullet \mathbf{u}_f$

対象のベクトル　基底

他の成分が紛れてない，
周波数 f の成分を保証

Octaveでの検証


```
N = 8;
```
◀次元数（好きな数字を入れていい，10000とかはやめたほうがいいが）

```
t = f = [0:N-1];
```
◀t と f に0~N-1の整数を順に入れる

```
U = exp(-i*2*pi*t'*f/N);
```
◀t を列 f を行とし，$\mathbf{u}_f$ を作成

```
round(abs(U'*U))
```
◀abs と round はなくてもよいが，
複素数と小数点で見にくくなるのでカット

```
1  #unit_vectors_DFT
2  N = 8;
3  t = f = [0:N-1];
4  U = exp(i*2*pi*t'*f/N);
5  round(abs(U'*U))
```

コマンドウィンドウ

```
>> unit_vectors_DFT

ans =

   8   0   0   0   0   0   0   0
   0   8   0   0   0   0   0   0
   0   0   8   0   0   0   0   0
   0   0   0   8   0   0   0   0
   0   0   0   0   8   0   0   0
   0   0   0   0   0   8   0   0
   0   0   0   0   0   0   8   0
   0   0   0   0   0   0   0   8

>>
```

※ i に既に整数等を代入している場合には clear しないと虚数単位にはならない

◀直交性が確認できる.

ちなみにUは…

U =

$\mathbf{u}_0$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_6$	$\mathbf{u}_7$
1.00000 + 0.00000i	1.00000 + 0.00000i	1.00000 + 0.00000i	1.00000 + 0.00000i	1.00000 + 0.00000i	1.00000 + 0.00000i	1.00000 + 0.00000i	1.00000 + 0.00000i
1.00000 + 0.00000i	0.70711 + 0.70711i	0.00000 + 1.00000i	-0.70711 + 0.70711i	-1.00000 + 0.00000i	-0.70711 - 0.70711i	-0.00000 - 1.00000i	0.70711 - 0.70711i
1.00000 + 0.00000i	0.00000 + 1.00000i	-1.00000 + 0.00000i	-0.00000 - 1.00000i	1.00000 - 0.00000i	0.00000 + 1.00000i	-1.00000 + 0.00000i	-0.00000 - 1.00000i
1.00000 + 0.00000i	-0.70711 + 0.70711i	-0.00000 - 1.00000i	0.70711 + 0.70711i	-1.00000 + 0.00000i	0.70711 - 0.70711i	0.00000 + 1.00000i	-0.70711 - 0.70711i
1.00000 + 0.00000i	-1.00000 + 0.00000i	1.00000 - 0.00000i	-1.00000 + 0.00000i	1.00000 - 0.00000i	-1.00000 + 0.00000i	1.00000 - 0.00000i	-1.00000 + 0.00000i
1.00000 + 0.00000i	-0.70711 - 0.70711i	0.00000 + 1.00000i	0.70711 - 0.70711i	-1.00000 + 0.00000i	0.70711 + 0.70711i	-0.00000 - 1.00000i	-0.70711 + 0.70711i
1.00000 + 0.00000i	-0.00000 - 1.00000i	-1.00000 + 0.00000i	0.00000 + 1.00000i	1.00000 - 0.00000i	-0.00000 - 1.00000i	-1.00000 + 0.00000i	-0.00000 + 1.00000i
1.00000 + 0.00000i	0.70711 - 0.70711i	-0.00000 - 1.00000i	-0.70711 - 0.70711i	-1.00000 + 0.00000i	-0.70711 + 0.70711i	-0.00000 + 1.00000i	0.70711 + 0.70711i

N 次元の離散フーリエ変換

N個の値を持つデータ列$\boldsymbol{v}$

$$\mathbf{v} = \begin{Bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{N-1} \end{Bmatrix}$$ に対して,

$$\mathbf{u}_f = \begin{Bmatrix} \exp\left(-\mathrm{i}\frac{2\pi}{N} f \times 0\right) \\ \exp\left(-\mathrm{i}\frac{2\pi}{N} f \times 1\right) \\ \vdots \\ \exp\left(-\mathrm{i}\frac{2\pi}{N} f (N-1)\right) \end{Bmatrix}$$

との内積を
$f = 0, 1, \ldots, N\text{-}1$
と順次取っていけば,

$$\mathbf{v} = \frac{1}{N} \sum_{f=0}^{N-1} (\mathbf{v} \cdot \mathbf{u}_f) \mathbf{u}_f$$ と, 分解できる.

$\boldsymbol{u}_f$に対応する強度に変換する.

離散フーリエ変換（DFT）

フーリエ積分にみる関係性

f_gに関するフーリエ積分

この部分が
フーリエ変換$F_g(f)$

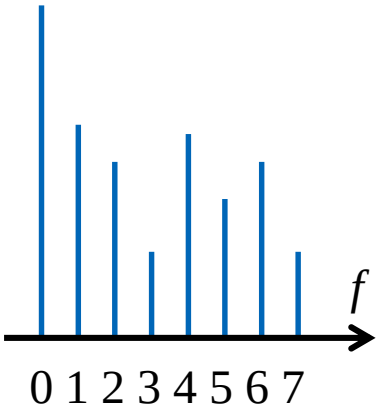
$$f_g(x)=\int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f_g(t)\exp(-\mathrm{i}\,2\pi f t)\,\mathrm{d}t\right)\exp(\mathrm{i}\,2\pi f x)\,\mathrm{d}f$$

fを1つの値に固定すると，
基底関数になる

関数の内積（無限区間での積分）
▸ ベクトルの内積 に対応

$$\mathbf{v}=\frac{1}{N}\sum_{f=0}^{N-1}(\mathbf{v}\cdot\mathbf{u}_f)\mathbf{u}_f$$

内積値を
fの番号順に並べる



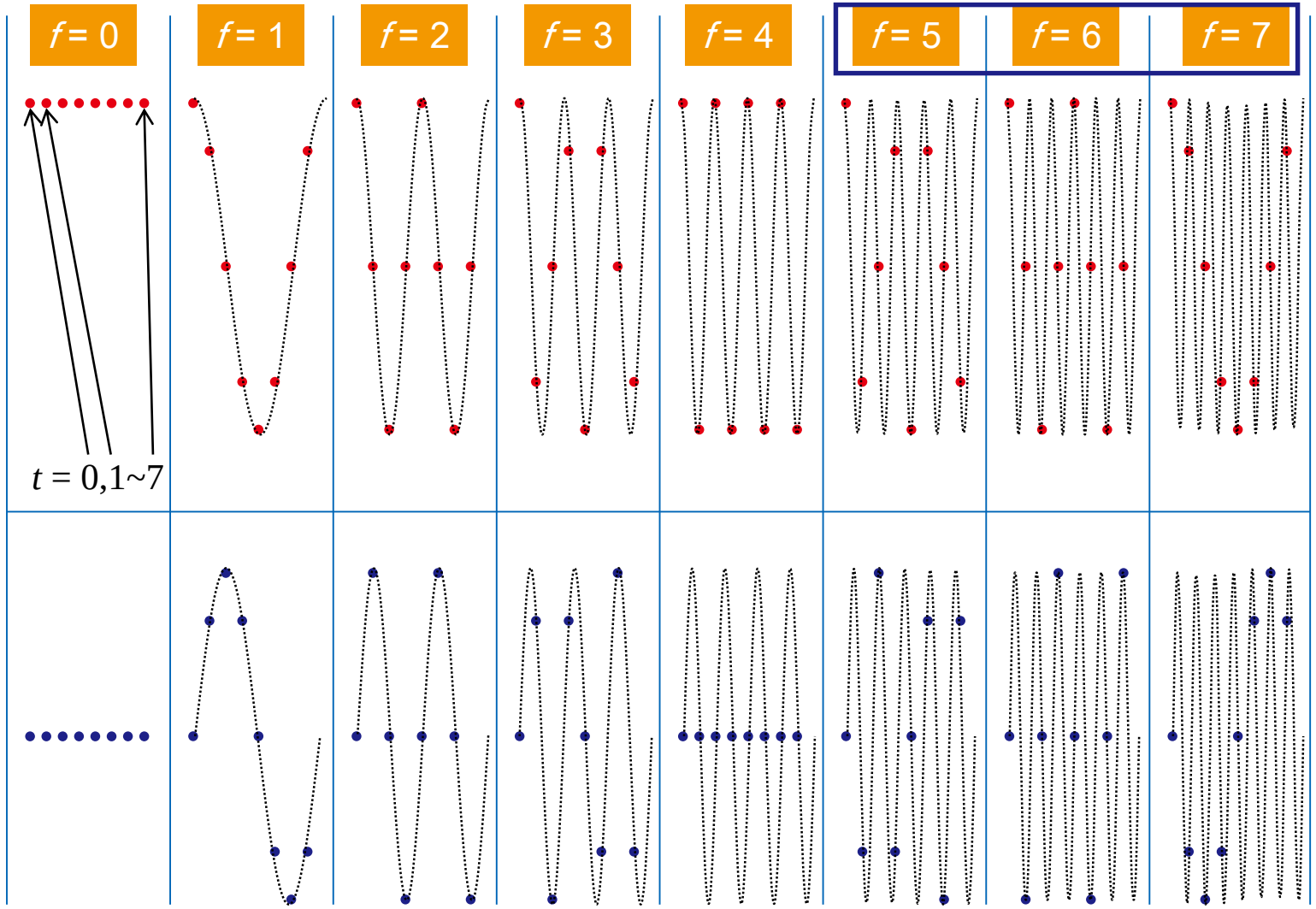
周波数ごとの強度が見える

$N = 8$ の場合

再現できない ▸ シャノンのサンプリング定理

$$\exp\left(-\mathrm{i}\,\frac{2\pi f t}{N}\right)$$

に関して

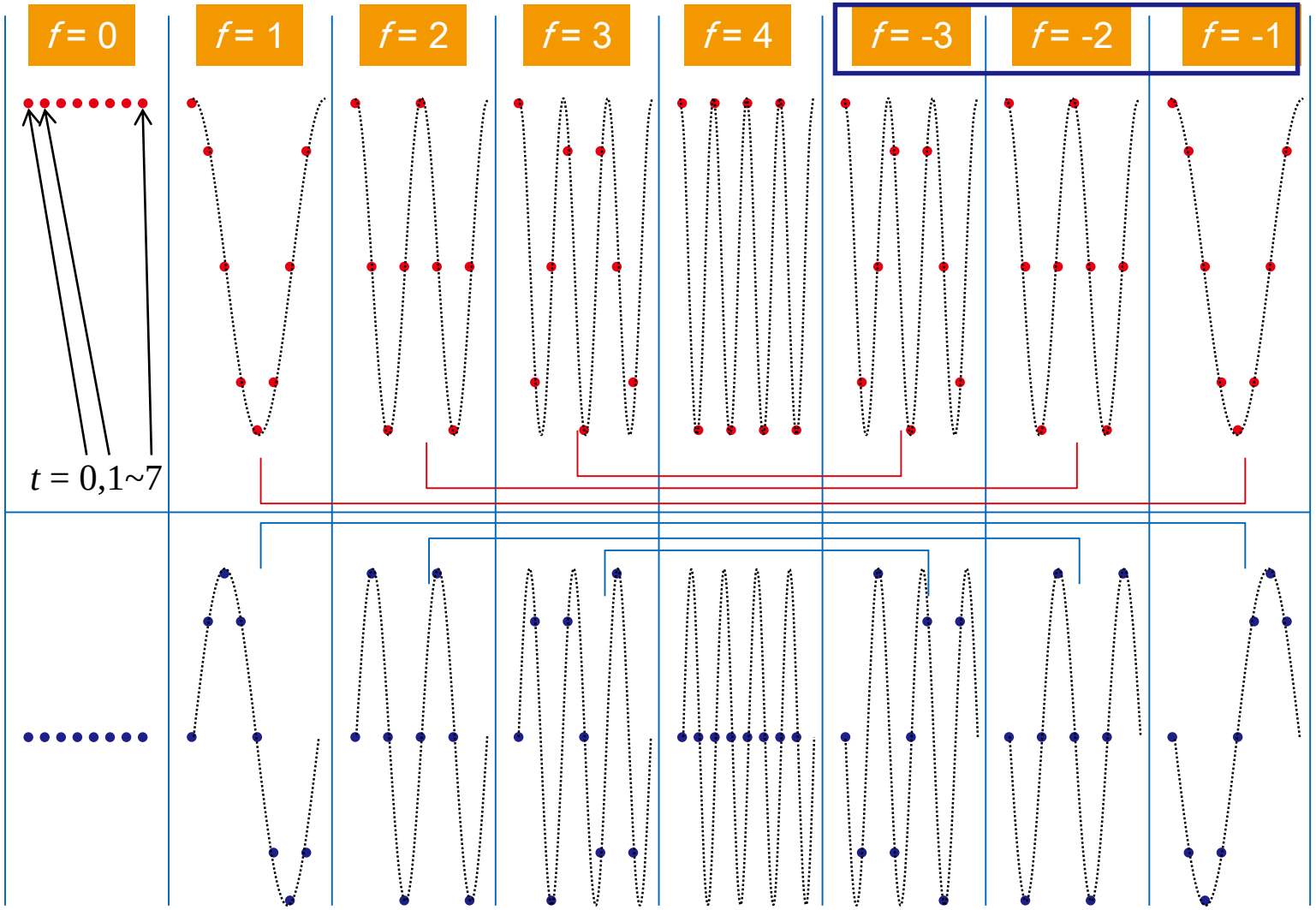


$N = 8$ の場合

結果的に符号反転の状態となる.

$$\exp\left(-\mathrm{i}\,\frac{2\pi ft}{N}\right)$$

に関して



高速フーリエ変換

対称性を利用して効率的に係数を計算する方法である

例えば

$a \cdot x + b \cdot x$	掛け算2回，足し算1回
$(a + b) \cdot x$	掛け算1回，足し算1回

因数分解で
掛け算の回数が減る！

データ数が2^nの場合

	離散的フーリエ変換	高速フーリエ変換
足し算	$2^{2n} - (n+2) \cdot 2^{n-1}$	$(n-1) \cdot 2^n + 1$
引き算	$n \cdot 2^{n-1}$	$2^n - 1$
掛け算	$2^{2n} - (n+1) \cdot 2^n$	$(n-2) \cdot 2^n + 2$

高速フーリエ変換

対称性を利用して効率的に係数を計算する方法である

$W_4 = \exp\left(-\mathrm{i}\frac{2\pi}{N}\right)$ として，$N = 4$の場合を例として考える

▲以下，下付き4を省略

$$\begin{Bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{Bmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{pmatrix} \begin{Bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{Bmatrix}$$

4回の掛け算
×4行

厳密には
1倍，-1倍，虚数単位倍は，
和，差，実虚入れ替えなので
掛け算ではないけど．

$W^4 = W^0 = 1$
$W^2 = -1$

なので，

高速フーリエ変換

$$\begin{Bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{Bmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & -1 & -W^1 \\ 1 & -1 & 1 & -1 \\ 1 & -W^1 & -1 & W^1 \end{pmatrix} \begin{Bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{Bmatrix}$$

▼各行列で2入力のペアを作る

$$\begin{Bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{Bmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -W^1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{Bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{Bmatrix}$$

2個の行列 × 2回の掛け算 × 4行

計算回数が減ってない. .

高速フーリエ変換

対称性を利用して効率的に係数を計算する方法である

$W_8 = \exp\left(-\mathrm{i}\frac{2\pi}{N}\right)$ として，$N = 8$の場合を例として考える

▲以下，下付き8を省略

8回の掛け算

×8行

$$\begin{Bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \end{Bmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^8 & W^{10} & W^{12} & W^{14} \\ W^0 & W^3 & W^6 & W^9 & W^{12} & W^{15} & W^{18} & W^{21} \\ W^0 & W^4 & W^8 & W^{12} & W^{16} & W^{20} & W^{24} & W^{28} \\ W^0 & W^5 & W^{10} & W^{15} & W^{20} & W^{25} & W^{30} & W^{35} \\ W^0 & W^6 & W^{12} & W^{18} & W^{24} & W^{30} & W^{36} & W^{42} \\ W^0 & W^7 & W^{14} & W^{21} & W^{28} & W^{35} & W^{42} & W^{49} \end{pmatrix} \begin{Bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{Bmatrix}$$

この行列を分解すると．．．

高速フーリエ変換

$$\begin{Bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \end{Bmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & W^1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & W^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & W^3 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -W^1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -W^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -W^3 \end{pmatrix} \begin{pmatrix} I_2 & I_2 & \mathbf{0}_2 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & I_2 & W^2 I_2 \\ I_2 & -I_2 & \mathbf{0}_2 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & I_2 & -W^2 I_2 \end{pmatrix} \begin{pmatrix} I_4 & I_4 \\ I_4 & -I_4 \end{pmatrix} \begin{Bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{Bmatrix}$$

変形は
各自追って

2回の掛け算
×8行

2回の掛け算
×8行

2回の掛け算
×8行

8×8＝64 回 ▶ 3×2×8＝48 回に減少

高速フーリエ変換

一般に,

$2^n \times 2^n$行列 ▶ 分解 ▶ n個の$2^n \times 2^n$行列

（ざっくりした）
掛け算の回数

※厳密には1倍や-1倍，i倍は計算負荷が軽いのだが概算で

2^n回 × 2^n行

n個の行列 × 2回 × 2^n行

計算量

$O(N^2)$ ▶ $O(N \log N)$ に減小!!

例えば1024個のデータなら，100万回程度の計算を，1万回以下にする