2021年5月31日作成（2022年6月15日修正）

音楽情報処理講義補助資料（関西学院大学工学部）

作成：森鈴果

Google Colaboratoryを用いた

NMFマニュアル

参考：**https://thinkit.co.jp/article/17411**

https://yokaze.github.io/2019/08/12/

1. ライブラリのインポート

参考：https://yokaze.github.io/2019/08/12/

前回インストールしていない以下をインストール。

```
from matplotlib import pyplot as pl
from matplotlib.gridspec import GridSpec
```

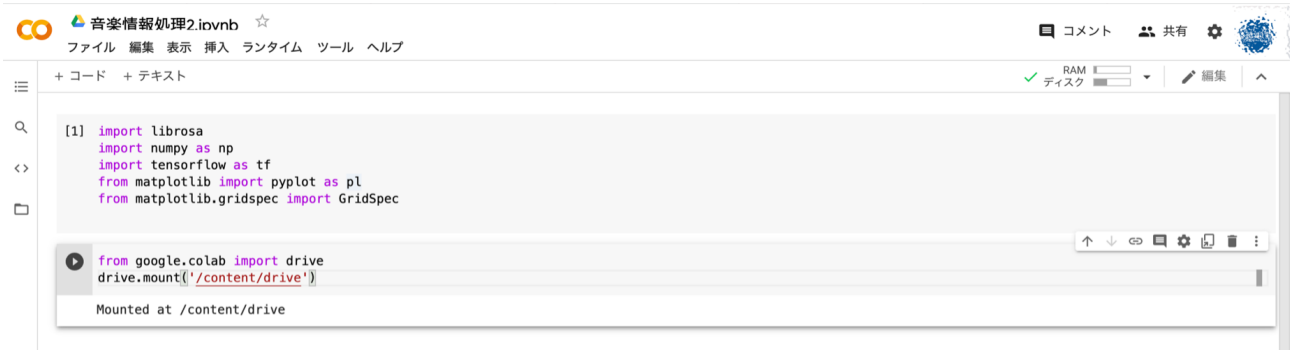
前回インストールした分は除く。

セルの部分に貼付して実行。(実行方法は前回分を参照)

2. GoogleDriveとの連携

前回と同様。

```
from google.colab import drive
drive.mount('/content/drive')
```



3. 音楽ファイルの読み込み

今回使用：music_test.mp3

twinkletwinklelittlestar.mp

前回と同様に、Librosaライブラリで読み込み。(視聴方法は前回参照)

```
y, sr = librosa.load('drive/My Drive/?????.mp3', mono=True)
```

＊もし音源の長さが30秒以上だと学習に時間がかかるので、

以下のコードを入力して、先頭30秒だけをyとしてください。

```
nsecond = 30
y= y[:sr * nsecond]
```

読み込んだ音楽ファイルをフーリエ変換し絶対値をとったものを、

平均で割る。(yの平均が1になるよう正規化)

```
y_stft = abs(librosa.stft(y).T)
y_stft /= y_stft.mean()
```

4.NMF

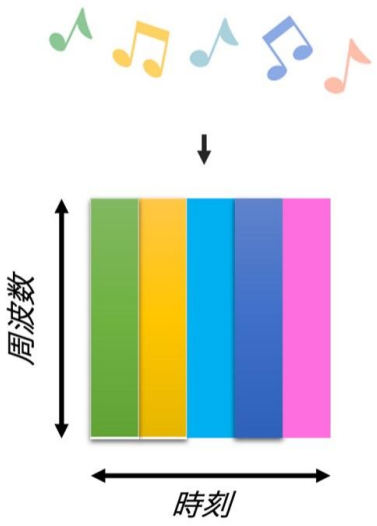
参考：https://www.sp.ipc.i.u-tokyo.ac.jp/~saruwatari/SP-Grad2018_02.pdf

NMFとは

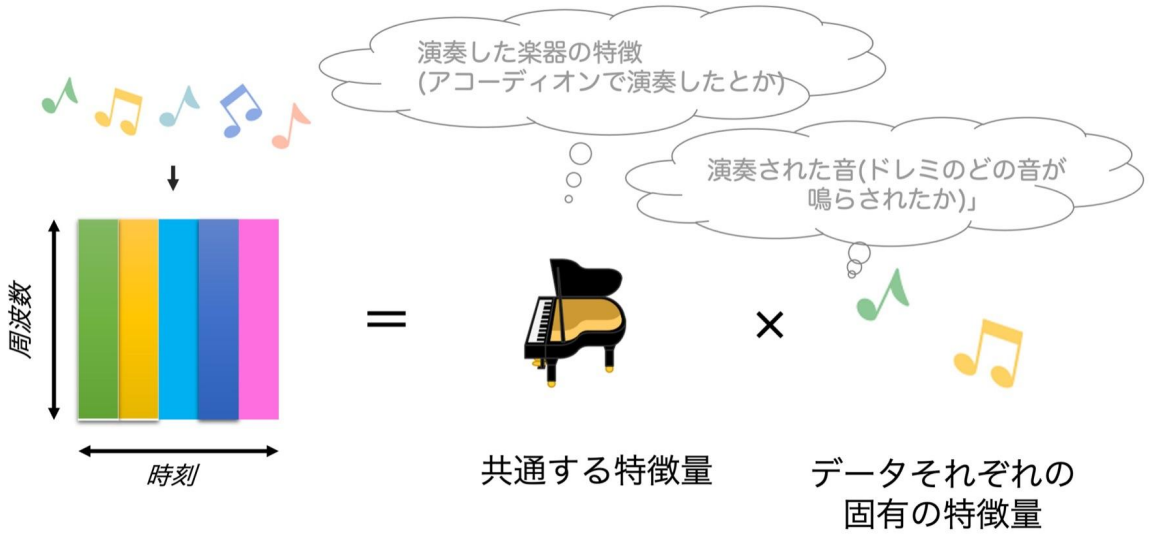
目的：音源データから、「演奏した楽器の特徴(アコーディオンで演奏したとか)」と「演奏された音(ドレミのどの音が鳴らされたか)」を分割する。

考え方

下の図のようにlibrosaで読み込まれた音楽ファイル（3章）は、縦軸周波数、横軸時刻の形状である。



今回は音楽ファイルの情報を、「全時刻に共通する音の特徴量」と「時刻ごとで異なる音程固有の特徴量」に分割することで、「演奏した楽器の特徴(アコーディオンで演奏したとか)」と目的の演奏された音(ドレミのどの音が鳴らされたか)を調べることを考える。



行列の計算方法から、下の図のように音楽ファイルは行列式によって2つの特徴量に分解すると「①全時刻に共通する音の特徴量」と「②時刻ごとで異なる音程固有の特徴量」に分解できそう。(①と②は非負の行列)

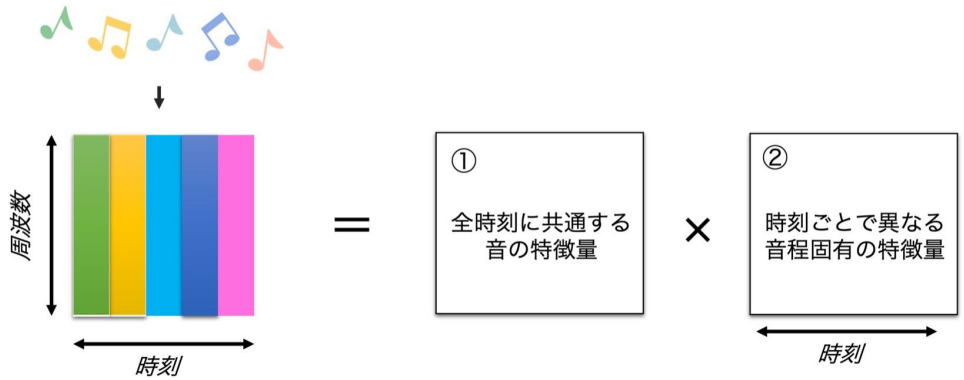
行列の計算方法

$$\begin{matrix} a_1 & a_2 \\ (a_3 & a_4) \\ a_5 & a_6 \end{matrix} \times \begin{pmatrix} b_1 & b_2 & b_3 \\ b_6 & b_7 & b_8 \end{pmatrix} = \begin{pmatrix} a_1b_1 + a_2b_6 & a_1b_2 + a_2b_7 & a_1b_3 + a_2b_8 \\ a_3b_1 + a_4b_6 & a_3b_2 + a_4b_7 & a_3b_3 + a_4b_8 \\ a_5b_1 + a_6b_6 & a_5b_2 + a_6b_7 & a_5b_3 + a_6b_8 \end{pmatrix}$$

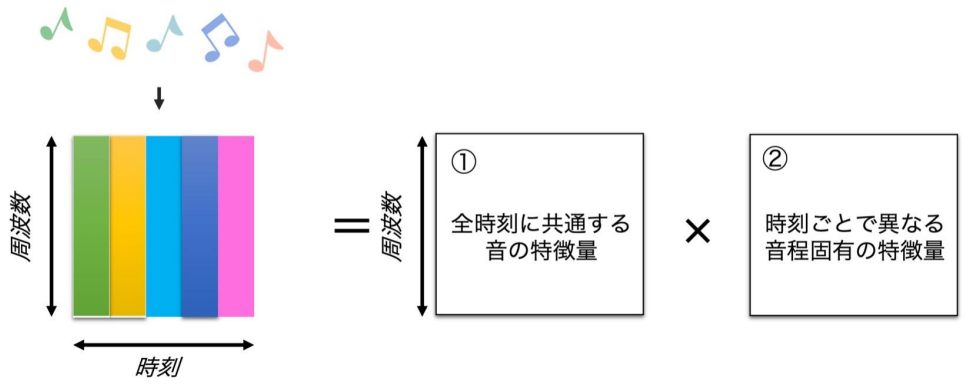
$$\begin{pmatrix} a_1b_1 + a_2b_6 & a_1b_2 + a_2b_7 & a_1b_3 + a_2b_8 \\ a_3b_1 + a_4b_6 & a_3b_2 + a_4b_7 & a_3b_3 + a_4b_8 \\ a_5b_1 + a_6b_6 & a_5b_2 + a_6b_7 & a_5b_3 + a_6b_8 \end{pmatrix} = \begin{matrix} a_1 & a_2 \\ (a_3 & a_4) \\ a_5 & a_6 \end{matrix} \times \begin{pmatrix} b_1 & b_2 & b_3 \\ b_6 & b_7 & b_8 \end{pmatrix}$$

行列の式にならって①と②に分解することで目的を満たす。

$$\begin{pmatrix} a_1b_1 + a_2b_6 & a_1b_2 + a_2b_7 & a_1b_3 + a_2b_8 \\ a_3b_1 + a_4b_6 & a_3b_2 + a_4b_7 & a_3b_3 + a_4b_8 \\ a_5b_1 + a_6b_6 & a_5b_2 + a_6b_7 & a_5b_3 + a_6b_8 \end{pmatrix} = \begin{matrix} a_1 & a_2 \\ (a_3 & a_4) \\ a_5 & a_6 \end{matrix} \times \begin{pmatrix} b_1 & b_2 & b_3 \\ b_6 & b_7 & b_8 \end{pmatrix}$$



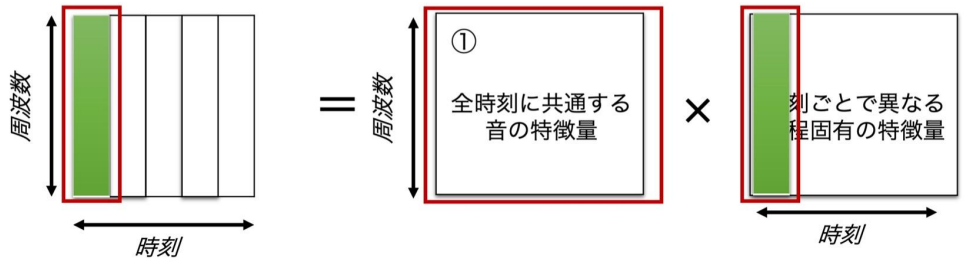
$$\begin{pmatrix} a_1b_1+a_2b_6 & a_1b_2+a_2b_7 & a_1b_3+a_2b_8 \\ a_3b_1+a_4b_6 & a_3b_2+a_4b_7 & a_3b_3+a_4b_8 \\ a_5b_1+a_6b_6 & a_5b_2+a_6b_7 & a_5b_3+a_6b_8 \end{pmatrix} = \begin{matrix} a_1 & a_2 \\ (a_3 & a_4) \\ a_5 & a_6 \end{matrix} \times \begin{pmatrix} b_1 & b_2 & b_3 \\ b_6 & b_7 & b_8 \end{pmatrix}$$



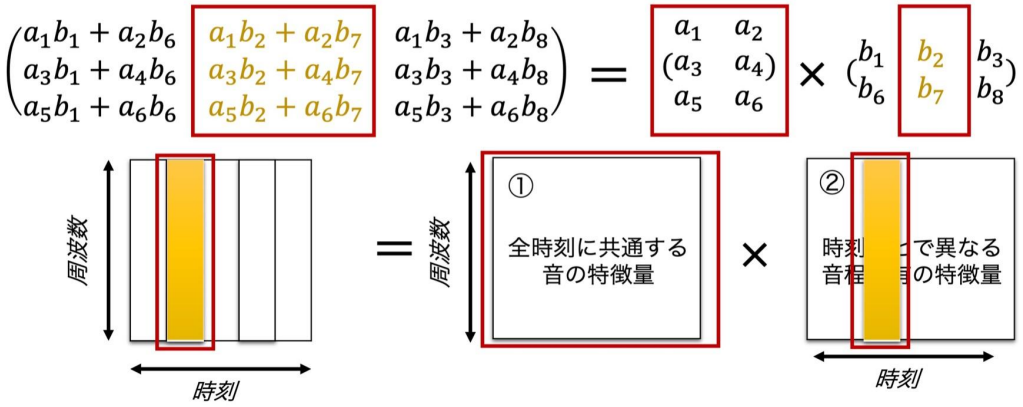
具体的には、今回の目的であるように行列①には「全時刻に共通する音の特徴量」、行列②には「時刻ごとで異なる音程固有の特徴量」が格納されるはず。

1番目になった音程（緑の音符）は、①の全行列と②の1列目をかけることで構成される。→②の1列目は音楽ファイルから全て時刻に共通する音の特徴量（①の行列）の影響を省いたもの。

$$\begin{pmatrix} a_1b_1+a_2b_6 & a_1b_2+a_2b_7 & a_1b_3+a_2b_8 \\ a_3b_1+a_4b_6 & a_3b_2+a_4b_7 & a_3b_3+a_4b_8 \\ a_5b_1+a_6b_6 & a_5b_2+a_6b_7 & a_5b_3+a_6b_8 \end{pmatrix} = \begin{matrix} a_1 & a_2 \\ (a_3 & a_4) \\ a_5 & a_6 \end{matrix} \times \begin{pmatrix} b_1 & b_2 & b_3 \\ b_6 & b_7 & b_8 \end{pmatrix}$$



2番目になった音程（黄色の音符）は、①の全行列と②の2列目をかけることで構成される。→②の2列目は音楽ファイルから全て時刻に共通する音の特徴量（①の行列）の影響を省いたもの。



これを解釈すると

「*NMF*は音データの集まりである行列Vの中に潜む 共通の特徴①、各要素の固有の特徴② を取り出す」となり、

今回の目的 (音源データから、「演奏した楽器の特徴(アコーディオンで演奏したとか)」と「演奏された音(ドレミのどの音が鳴らされたか)」を分割) を達成できるのではないか。

具体的には、

①には、例えば「音源データ(30_123.mp3)がアコーディオンからなっている」という情報とかが入るはず(基底スペクトル)。

②には、例えばどの音程の音が鳴ったかという情報が入るはず(各基底の「アクティベーション」)。

＊①の情報に②のi列目の情報を加えるから、①と②の行列は非負の値から構成する。

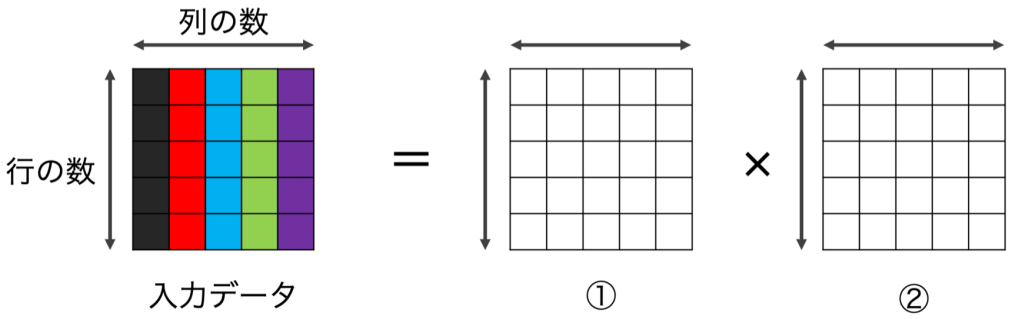
5. NMFの実装

データの準備

行列の計算方法から、

1. 入力データの行の数＝①の行の数
2. 入力データの列の数＝②の列の数
3. ①の列の数＝②の行の数

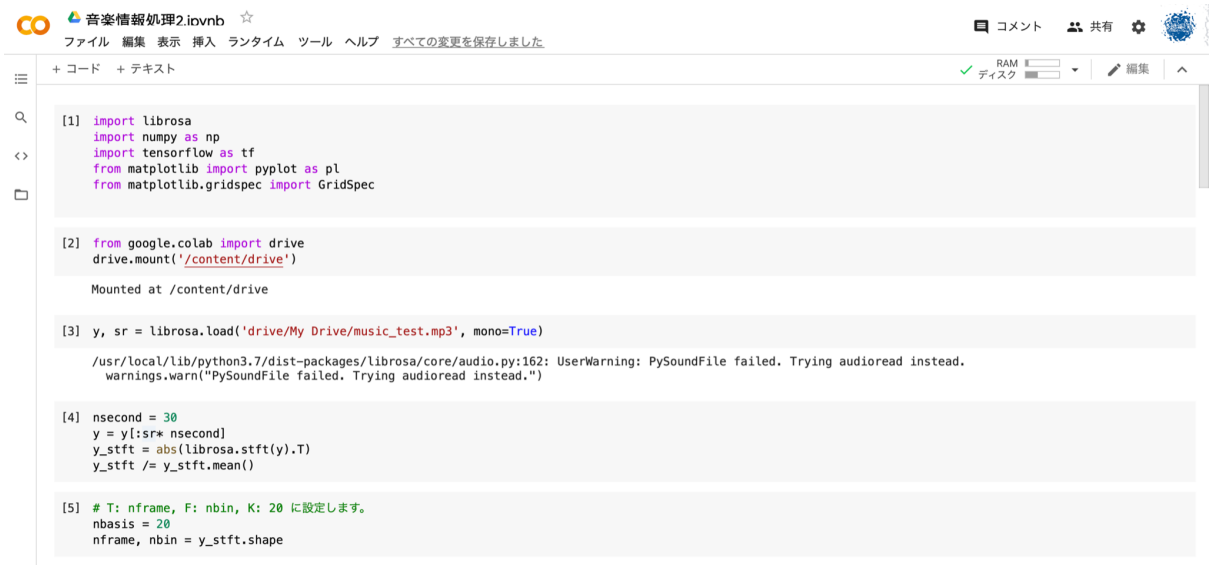
である必要がある。



よって、形状を以下で設定する。

ここでは、3. ①の列の数(②の行の数)は20とした。(これはハイパーパラメータで自分で好きなように設定できる。)

```
nbasis = 20
nframe, nbin = y_stft.shape
```



初期値の設定

計算する際に、音源ファイルの値に0が入っていると計算が上手くできないので予め `y_stft` に1e-10を加えている。
行列①(`lw`)、行列②(`lh`) はこれから求めていく(更新していく)ので初期値としてランダムな数が入っている。

```
history = []
x = tf.constant(y_stft + 1e-10)
lx = tf.constant(tf.math.log(x))
lw = tf.Variable(tf.random.normal([nframe, nbasis]))
lh = tf.Variable(tf.random.normal([nbasis, nbin]))
```

グラフのカラースケールを固定するため、1e-3 (-60dB) で足切り。

```
wh = tf.matmul(tf.exp(lw), tf.exp(lh)).numpy()
lw_display = tf.maximum(lw, tf.reduce_max(lw) +
np.log(1e-3)).numpy()
lh_display = tf.maximum(lh, tf.reduce_max(lh) +
np.log(1e-3)).numpy()
lwh_display = np.log(np.maximum(wh, wh.max() * 1e-3))
```

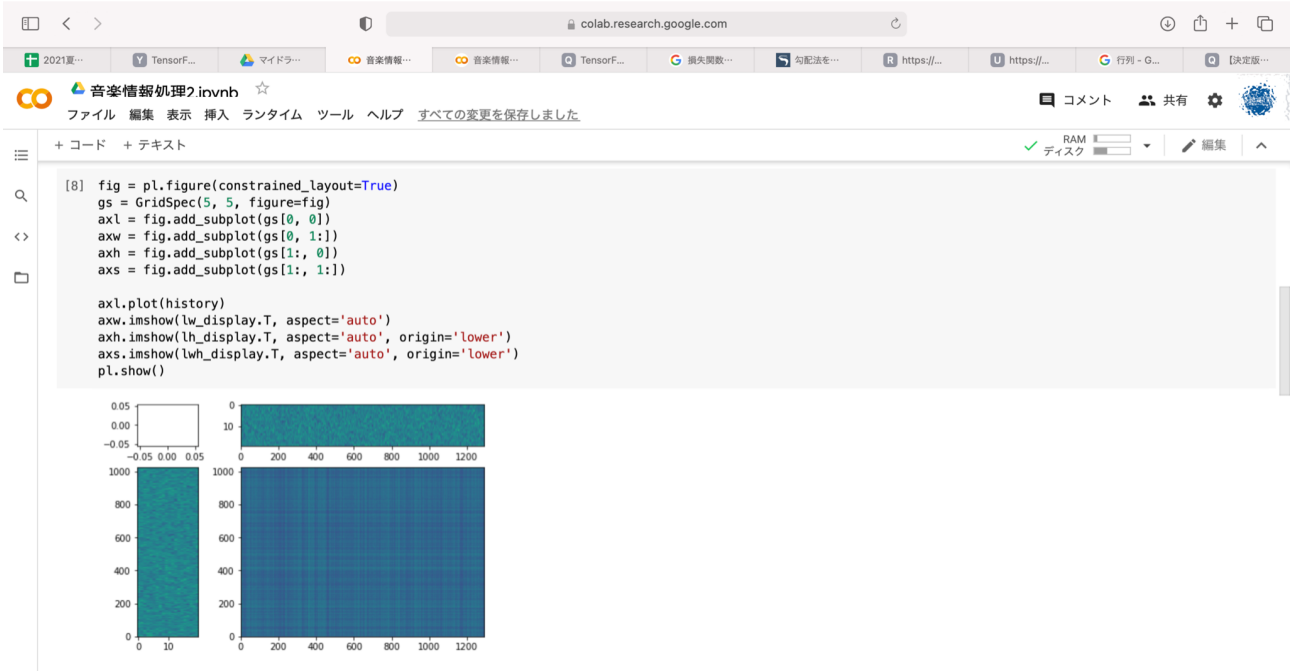
6. グラフの設定

ランダムな数で構成していた行列 ① (lw) と行列② (lh)、①と②の積 (lwとlhの積) をグラフで表示してみる。

```
fig = pl.figure(constrained_layout=True)
gs = GridSpec(5, 5, figure=fig)
axl = fig.add_subplot(gs[0, 0])
axw = fig.add_subplot(gs[0, 1:])
axh = fig.add_subplot(gs[1:, 0])
axs = fig.add_subplot(gs[1:, 1:])

axl.plot(history)
axw.imshow(lw_display.T, aspect='auto')
axh.imshow(lh_display.T, aspect='auto', origin='lower')
axs.imshow(lwh_display.T, aspect='auto', origin='lower')
pl.show()
```

ランダムな値が入っていることを確認。

グラフの図はどれも規則正しくない数値(色)になっている。



7. 学習

参考：https://ebi-works.com/deeplearning-5/

解説はコード下

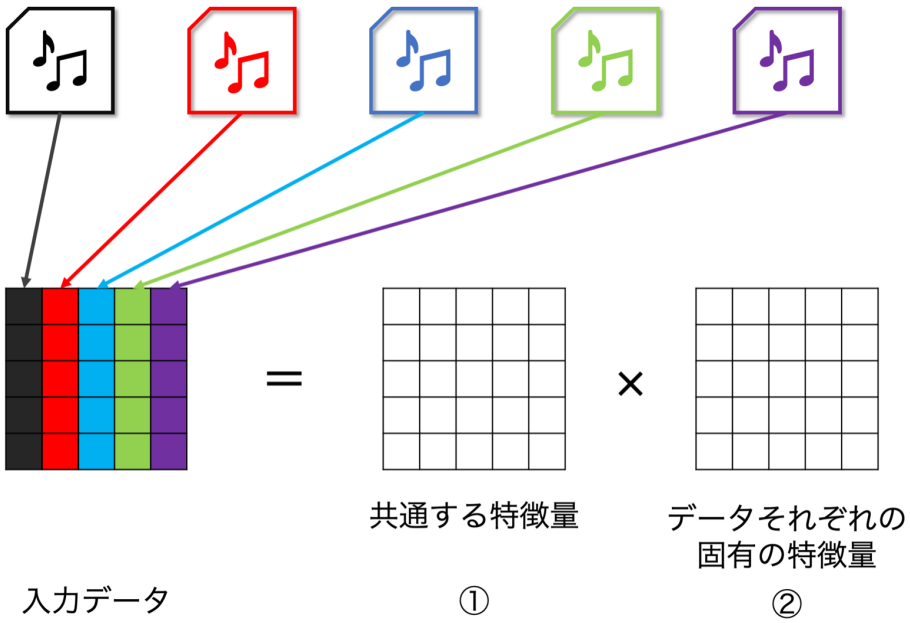
```
history = []
opt = tf.keras.optimizers.Adam(learning_rate=.1)

def loss_euc():
        wh = tf.matmul(tf.exp(lw), tf.exp(lh))
        return tf.reduce_sum((x - wh) ** 2)

def loss_idiv():
        lwh = tf.reduce_logsumexp(lw[:, :, None] + lh[None, :, :],
```

```
axis=1)
        return tf.reduce_sum(x * (lx - lwh - 1) + tf.exp(lwh))

for i in range(10000):
    print("何回目:")
    print(i)

    opt.minimize(loss_idiv, var_list=[lw, lh])
    history.append(np.log(loss_idiv().numpy()))

    wh = tf.matmul(tf.exp(lw), tf.exp(lh)).numpy()
    lw_display = tf.maximum(lw, tf.reduce_max(lw) +
np.log(1e-3)).numpy()
    lh_display = tf.maximum(lh, tf.reduce_max(lh) +
np.log(1e-3)).numpy()
    lwh_display = np.log(np.maximum(wh, wh.max() * 1e-3))
```

行列①と行列②をランダムな値から更新して、最適解を求める

4. NMF (NMF とは) から、

NMFでは、入力データを全ての音データに共通する特徴量(行列①) と 音データそれぞれの固有の特徴量(行列②) に分割することが目標だった。



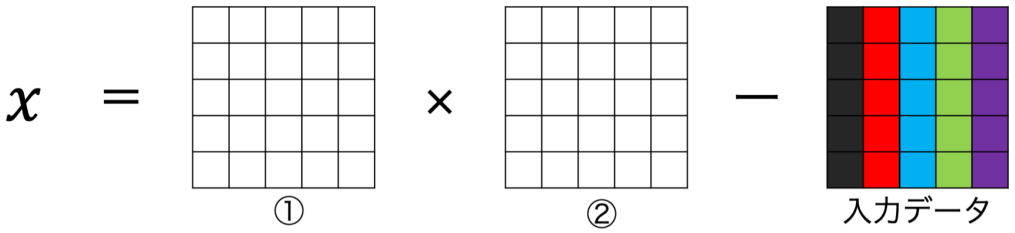
よって、①と②の積は入力データと等しくならなくてはならない。

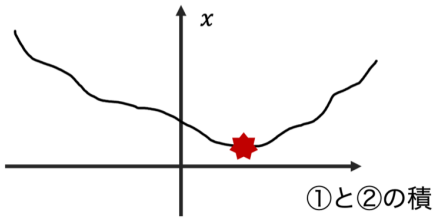
現段階でランダムな値が入っている①と②を、入力データと等しくなるよう更新していく。

「①と②の積は入力データと等しくならなくてはならない。」を

「①と②の積と入力データの差が0に近くあれば良い。」と考えると、

下図のようにxが最小な箇所を探せば良い。





：xが最小となる箇所
(①と②の積が入力データに最も近い箇所)

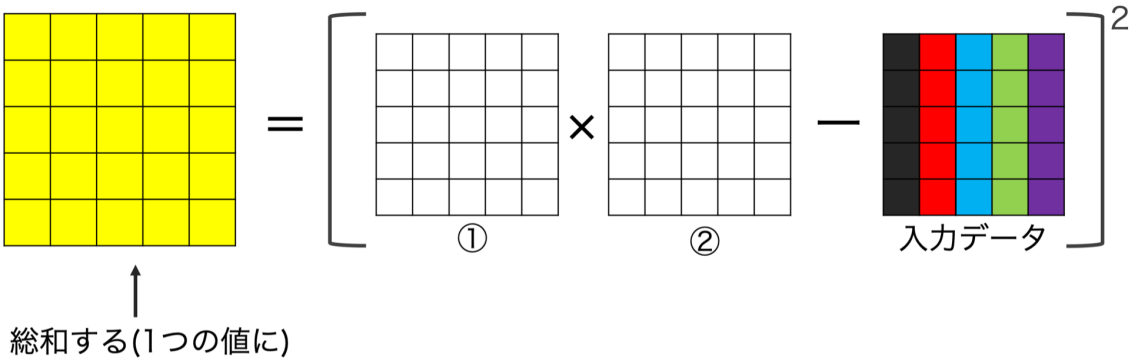
誤差の求め方（どちらかを選択)

#2乗誤差

差が負の値にならないようにするために2乗する。

(最小の値を、最も差が少ない場合としたい。)

この総和が0に近いものを探す。



$$x = (①と②の積 - 入力データ)^2$$

```
def loss_euc():
        wh = tf.matmul(tf.exp(lw), tf.exp(lh))
        return tf.reduce_sum((x - wh) ** 2)
```

#I-ダイバージェンス

2乗誤差をもう少し緩やかになるよう考えた誤差算出方法。

このxが0に近いものを探す。

$$x = ①と②の積 \times \log\frac{①と②の積}{入力データ} - (①と②の積 - 入力データ)$$

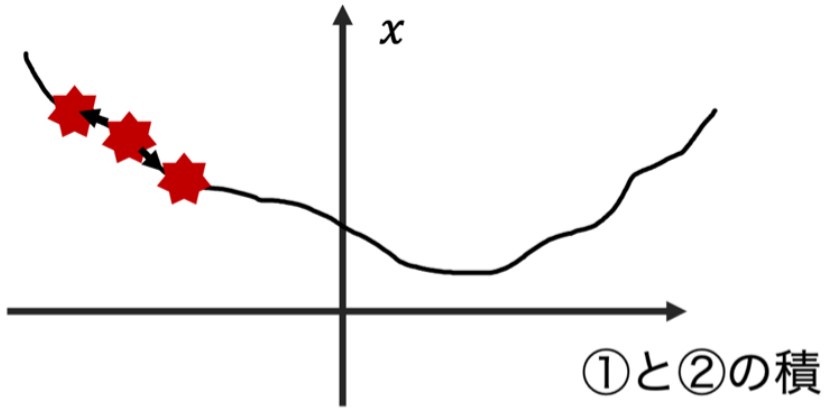
```
def loss_idiv():
        lwh = tf.reduce_logsumexp(lw[:, :, None] + lh[None, :, :],
axis=1)
        return tf.reduce_sum(x * (lx - lwh - 1) + tf.exp(lwh))
```

x (①と②の積と入力データの差) の更新と記録　(今回の誤差算出方法はI-ダイバージェンスの方)

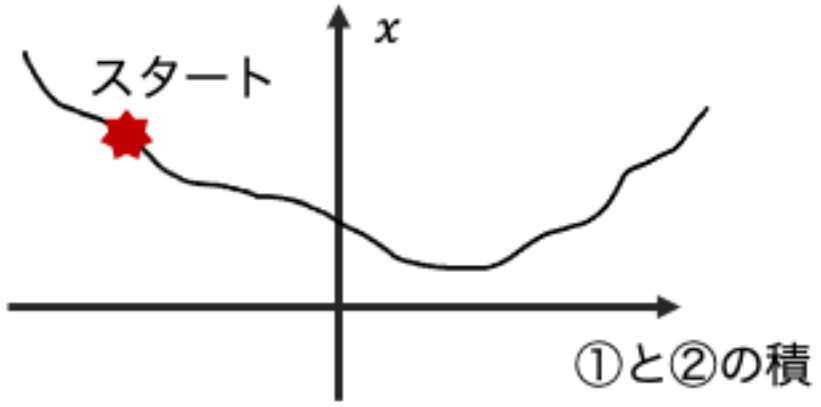
(2乗誤差にするには、`loss_idiv`を`loss_euc`にしてください。)

行列① (lw) と行列② (lh) の更新方法にAdamを用いる。

更新するとき、現在のxより小さいxの箇所 (下図の右側) に進むことで、よりx (①と②の積と入力データの差)を小さくできる。



更新方法の1つであるAdamは、更新するときにスタート時点のような傾きの大きい箇所では大きく更新し、ゴール地点の傾きの小さい箇所では小さく更新する手法。
今回はこのAdamを用いて更新する。
参考：https://qiita.com/omiita/items/1735c1d048fe5f611f80



$$h \leftarrow h + \frac{\partial x}{\partial(①と②の積)} \odot \frac{\partial x}{\partial(①と②の積)}$$

$$①と②の積 \leftarrow ①と②の積 - 学習係数 \times \frac{1}{\sqrt{h}} \odot \frac{\partial x}{\partial(①と②の積)}$$

今回の学習係数(`learning_rate` は0.1とする。)

```
opt = tf.keras.optimizers.Adam(learning_rate=.1)
opt.minimize(loss_idiv, var_list=[lw, lh])
```

今回の誤差(x)を記録する。

```
history.append(np.log(loss_idiv().numpy()))
```

グラフのカラースケールを固定するため、1e-3 (-60dB) で足切り。(4章と同様)

```
wh = tf.matmul(tf.exp(lw), tf.exp(lh)).numpy()
lw_display = tf.maximum(lw, tf.reduce_max(lw) +
np.log(1e-3)).numpy()
lh_display = tf.maximum(lh, tf.reduce_max(lh) +
np.log(1e-3)).numpy()
lwh_display = np.log(np.maximum(wh, wh.max() * 1e-3))
```

この作業を10000回繰り返す。

(13インチ MacBookPro：プロセッサ 2.8 GHz Core i7

macOS Big Sur(11.3) + TensorFlow 2.5での学習)

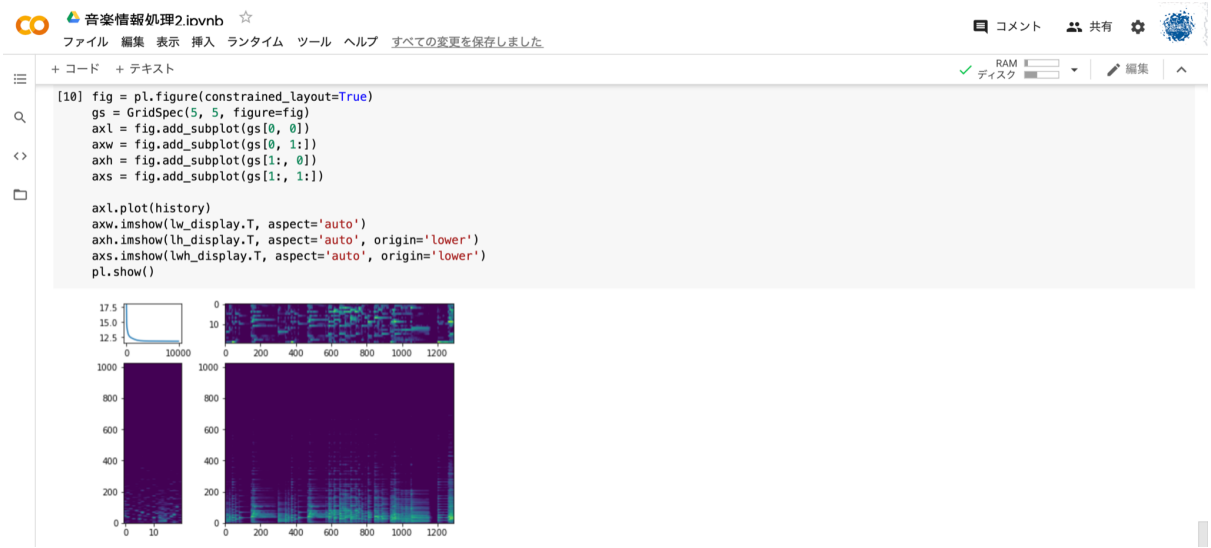
```
for i in range(10000):
```

8. 結果の出力

6章同様に出力。

```
fig = pl.figure(constrained_layout=True)
gs = GridSpec(5, 5, figure=fig)
axl = fig.add_subplot(gs[0, 0])
axw = fig.add_subplot(gs[0, 1:])
axh = fig.add_subplot(gs[1:, 0])
axs = fig.add_subplot(gs[1:, 1:])

axl.plot(history)
```

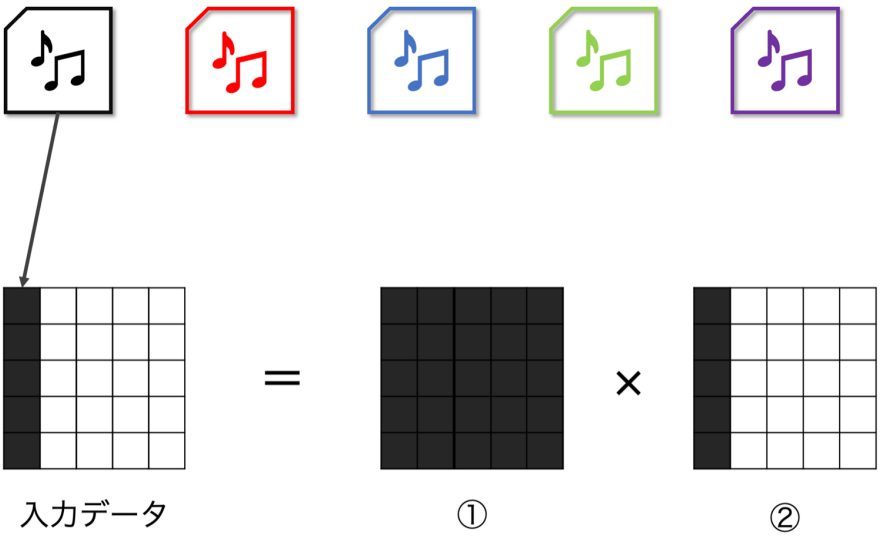
```
axw.imshow(lw_display.T, aspect='auto')
axh.imshow(lh_display.T, aspect='auto', origin='lower')
axs.imshow(lwh_display.T, aspect='auto', origin='lower')
pl.show()
```



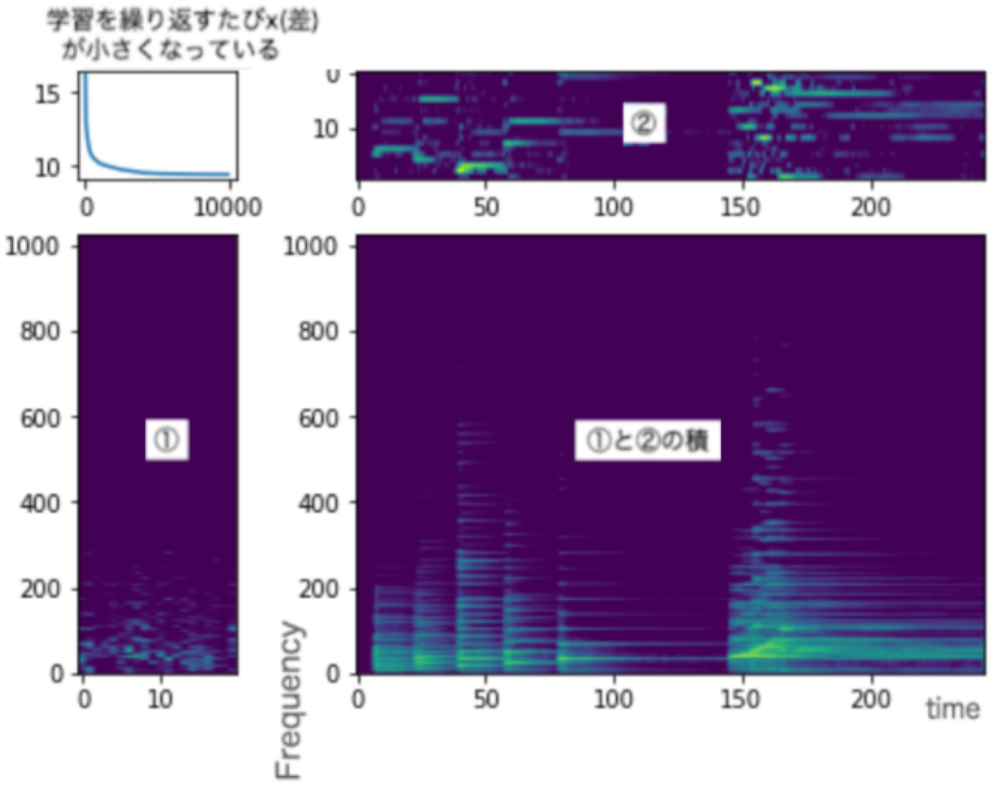
出力された図から、②は「演奏された音(ドレミのどの音が鳴らされたか)」が生成されている。

> 行列①：入力データを全ての音データに共通する特徴量（基底スペクトル。例えば音源データ(30_123.mp3)がアコーディオンからなっている）という情報。
> 行列②：音データそれぞれの固有の特徴量（各基底のアクティベーション。例えばどの音程の音が鳴ったかという情報）
> [参考：信号処理論特論 第2回 (5/1)]

x (行列①と行列②の積と入力データの差)も学習を繰り返すほど小さくなっている。



＊music_test.mp3（出力まで15分ほどかかる）

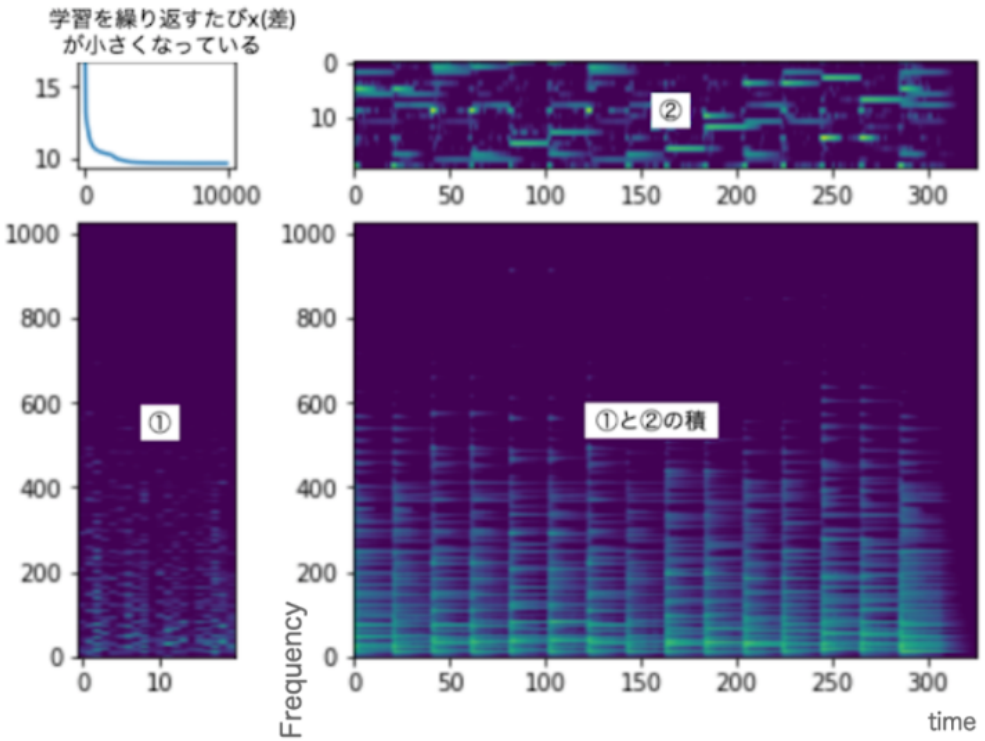


＊twinkletwinklelittlestar.mp3（出力まで20分ほどかかる）

```
for i in range(5000):
```

で5000回学習だと10分程度でできる。曲によって学習回数は試してみてください。

（下の 学習過程 参照）



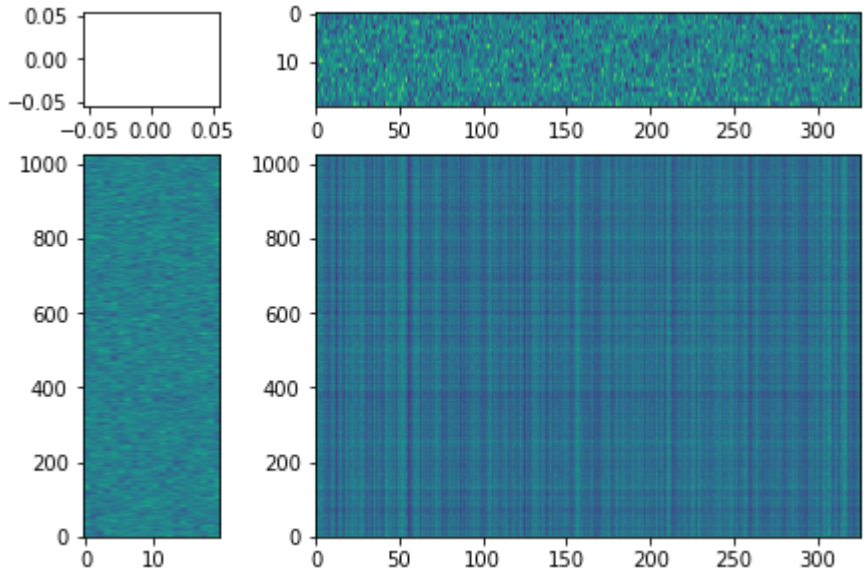
学習過程

今回は、学習（ 7. 学習 のコードの下部分）を10000回行っている。

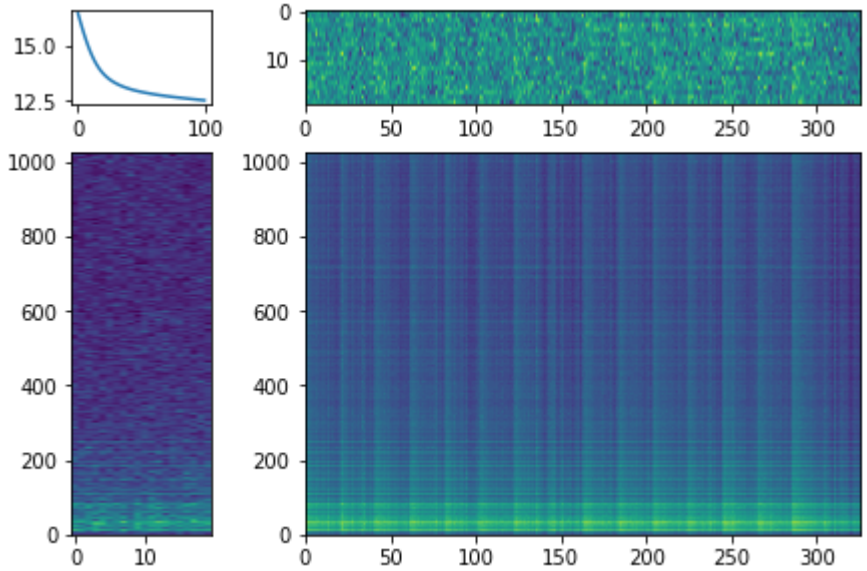
```
for i in range(10000):
```

その学習過程について確認する。

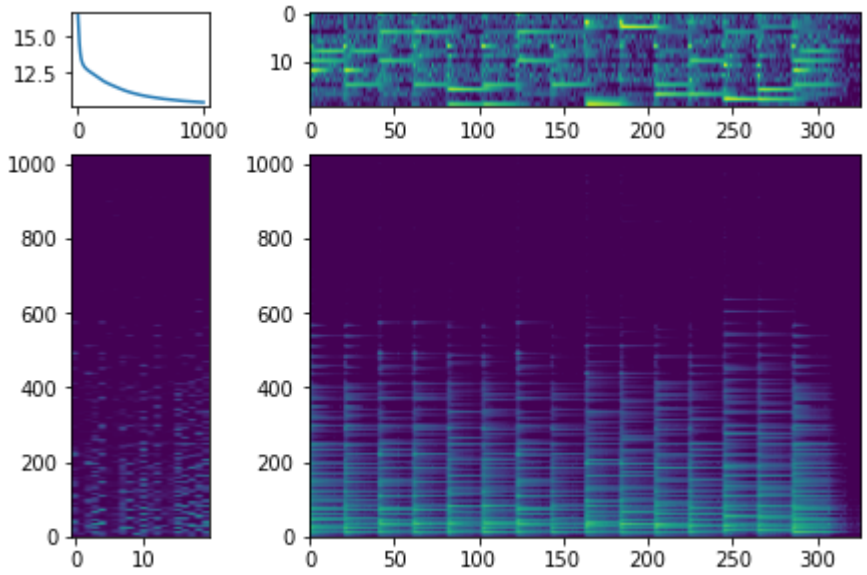
0回目（5. NMFの実装 初期値の設定 の時点）＊twinkletwinklelittlestar.mp3



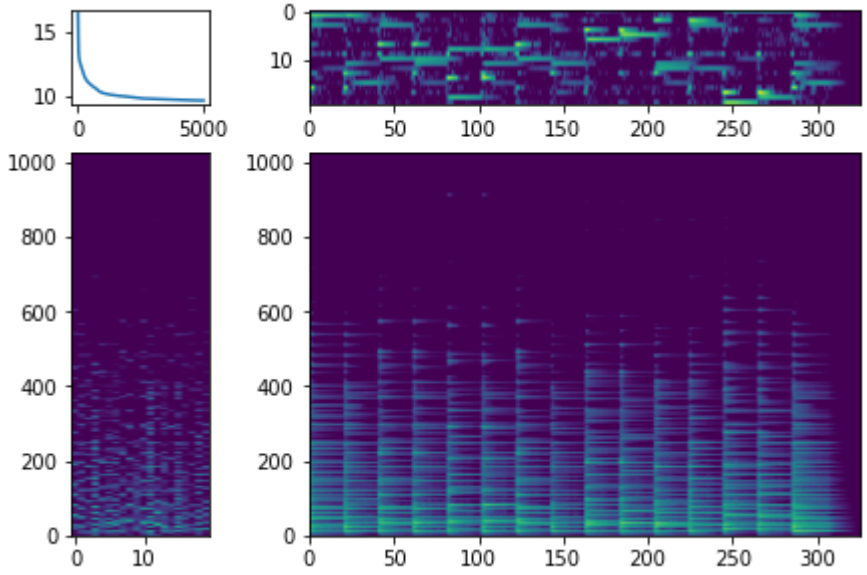
100回目



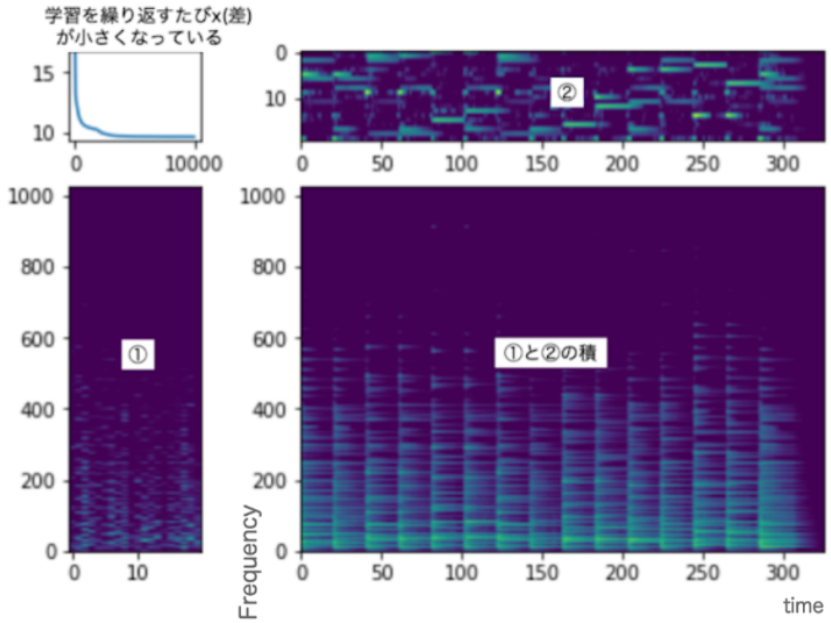
1000回目＊twinkletwinklelittlestar.mp3



5000回目＊[twinkletwinklelittlestar.mp3]



10000回目（再掲）＊[twinkletwinklelittlestar.mp3]



以上から、0回目→1000回目でだいたい②が可視化され始めていることがわかる。

学習の誤差（左上の図）を見ても、1000回目でだいぶ小さくなっていることから、

＊twinkletwinklelittlestar.mp3では、1000回程度の学習で十分な学習ができる。