# 第3回
# 線形代数 with Octave

2021.06.18（Fri.）

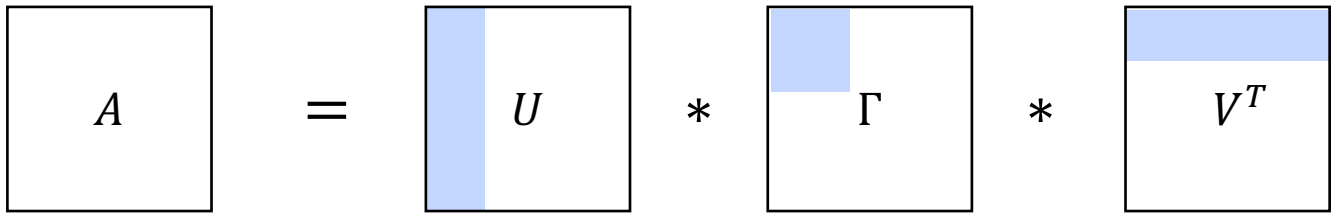
菅沼 雅徳，川越 喜晃

# 本日の講義内容

- 線形代数の知識＋Octaveで課題を解く
- 線形代数の基礎
- Octaveでの行列演算について

**今回の演習課題**

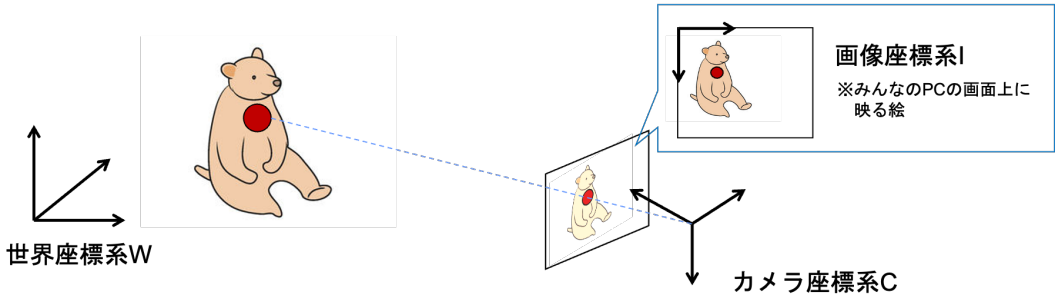
画像を読み込み，特異値分解＋低ランク近似＋近似後の画像表示を行うプログラム



# Why 線形代数？

## 現実空間の事象を扱うのに必要

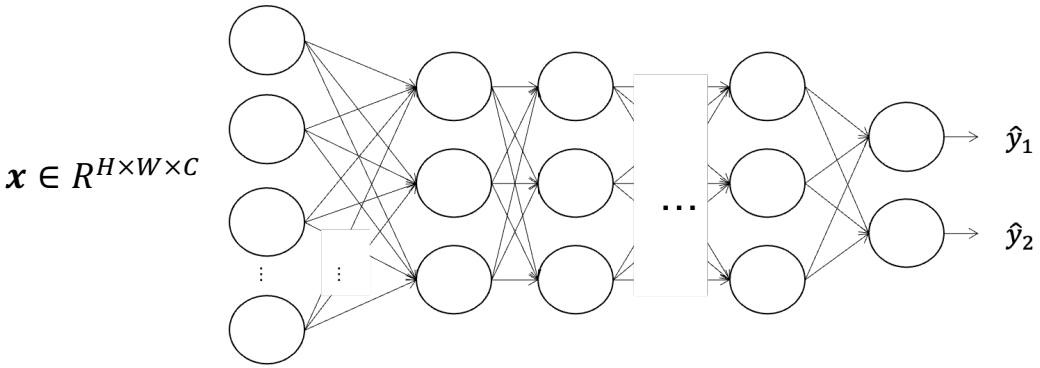
- コンピュータグラフィックス
- ゲーム，VR，AR
- 例）3次元にある物体を2次元画面に描画




## 多次元データを効率的に扱うのに必要

- 工学分野では必須
- データ解析，機械学習（≈人工知能）
- システムの制御
- 例）主成分分析，パターン認識

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$



# 線形代数の基礎：行列

- 多次元データを行列もしくはベクトルで表現
- それらデータの変換を行列演算で実施

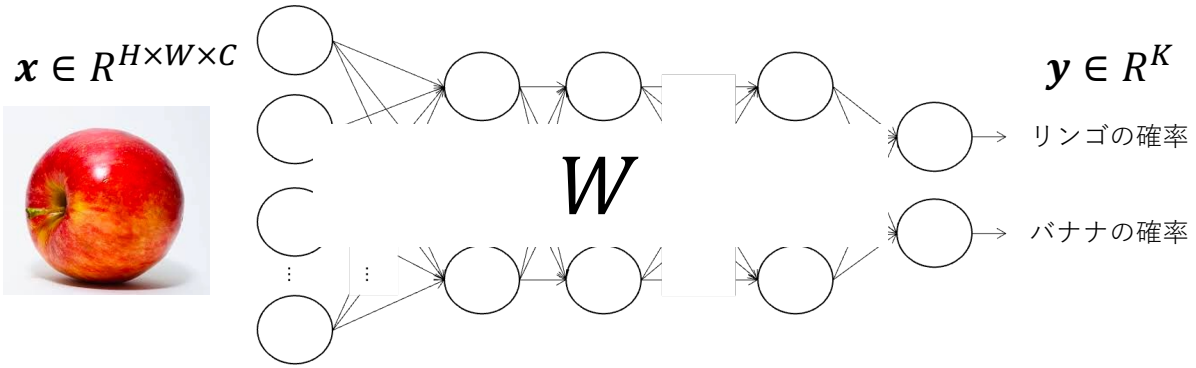
＝ **行列は写像もしくはデータそのもの**を表す

＝ 行列の特性を知ることが大事（ランク，固有値，対角化, ...）

例）$H \times W$のサイズの白黒画像

$$\boldsymbol{x} = R^{H \times W} = \overbrace{\begin{bmatrix} 3 & \cdots & 128 \\ \vdots & \ddots & \vdots \\ 32 & \cdots & 42 \end{bmatrix}}^{W} \Big\} H$$

例）ニューラルネットワークによる画像認識　$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$



# 行列 with Octave

## 行列の定義方法

- 自分で適当に行列を生成
- 外部データをOctaveの関数で読み込む
- Octaveで用意されている関数の使用（次ページ）

```
# 適当に生成
array = [1,2; 3,4]
array = [1,2,3; 4,5,6; 7,8,9]

# 外部データから読み込み（例．画像）
array = imread('lena.jpeg');
size(array)
```

```
array =

   1   2
   3   4

array =

   1   2   3
   4   5   6
   7   8   9

ans =

   225   225     3
```

# Octaveで使用可能な行列生成

## 単位行列

```
I = eye(3, 3)
```

```
I =

Diagonal Matrix

   1   0   0
   0   1   0
   0   0   1
```

## ゼロ行列

```
Z = zeros(3, 3)
```

```
Z =

   0   0   0
   0   0   0
   0   0   0
```

## 全要素1の行列

```
O = ones(3, 3)
```

```
O =

   1   1   1
   1   1   1
   1   1   1
```

## 乱数行列（[0.0, 1.0]の範囲）

```
R = rand(3, 3)
```

```
R =

   0.4632   0.9324   0.9216
   0.6951   0.4778   0.5429
   0.8117   0.9523   0.5521
```

## 乱数行列（平均0, 分散1の正規分布）

```
N = randn(3, 3)
```

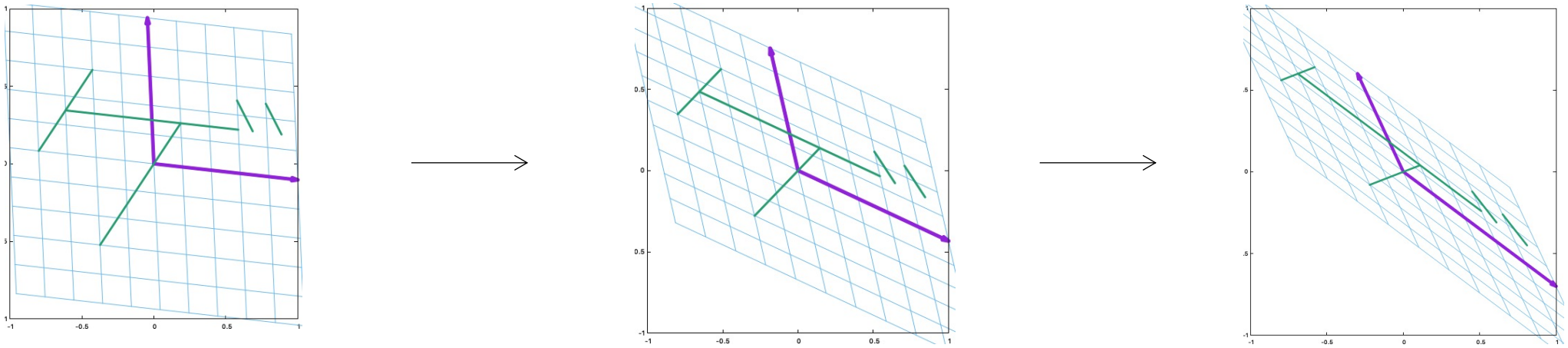
```
N =

  -1.843407  -1.409346  -0.137906
  -0.707010   0.014794   3.006673
   0.817805   1.868189  -1.606595
```

# 線形代数の基礎：行列による写像

- $n$次元ベクトル$\boldsymbol{x}$に$m \times n$行列$A$をかけると，$m$次元ベクトル$\boldsymbol{y} = A\boldsymbol{x}$が得られる
  ＝ 行列$A$によってベクトル$\boldsymbol{x}$を別のベクトル$\boldsymbol{y}$に写像（＝変換）

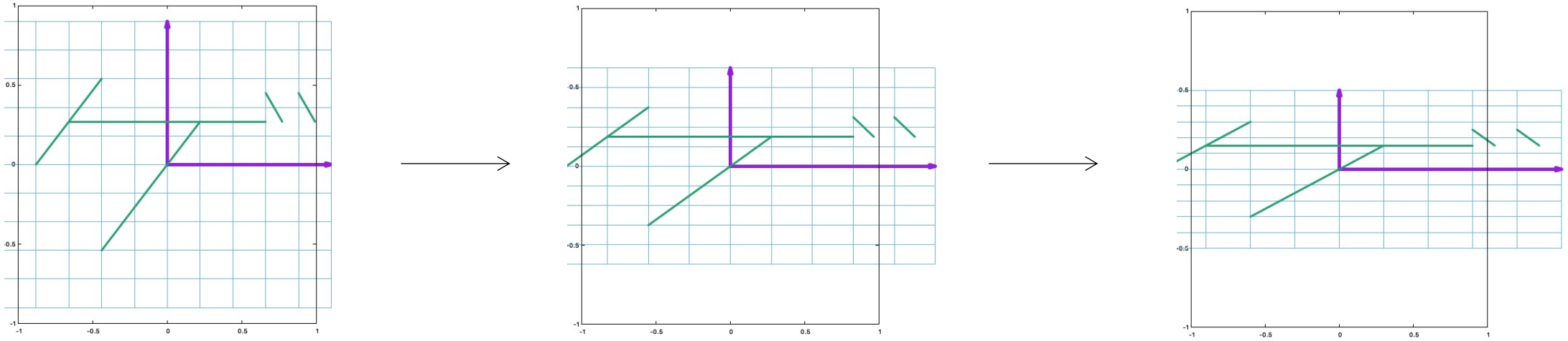
- 空間内の全てのベクトルが同じように行列によって変換されるため，
  空間全体が変形する（下図）

例）$A = \begin{bmatrix} 1 & -0.3 \\ -0.7 & 0.6 \end{bmatrix}$ による空間全体の変化

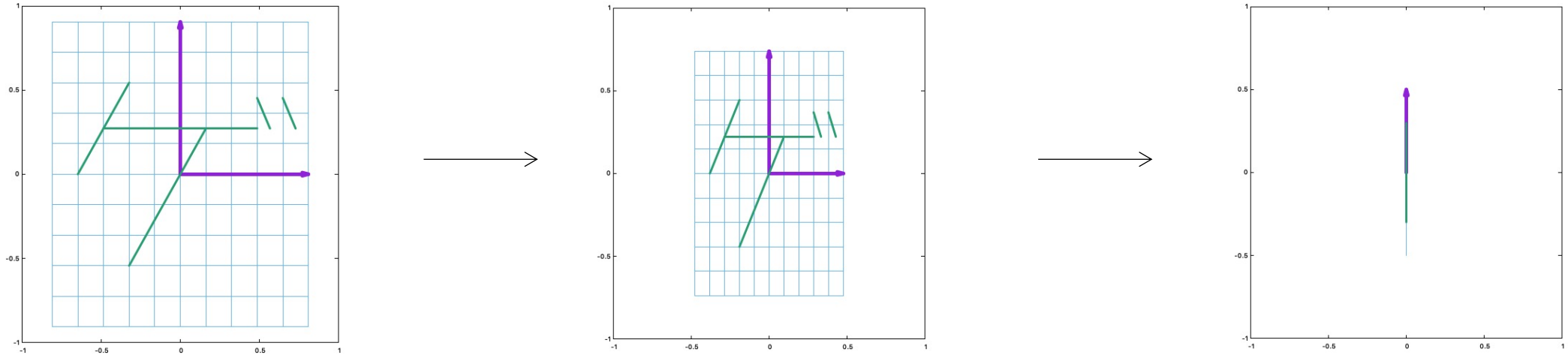


# 線形代数の基礎：行列による写像

例）$A = \begin{bmatrix} 1.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ による空間全体の変化（対角行列による写像は空間を伸縮させるだけ）



例）$A = \begin{bmatrix} 0 & 0 \\ 0 & 0.5 \end{bmatrix}$ による空間全体の変化（下記のように一方の次元が潰れる＝ランクが減少，逆行列が求まらない）



# 行列による写像 w/ Octave

## 行列積

- 演算子「*」で実行可能
- 行列およびベクトルの転置は「**変数.'**」

```
# y=Ax
x = [1,2,3]
A = [4,5,6; 7,8,9]
y = A * x.'
```

```
x =

   1   2   3

A =

   4   5   6
   7   8   9

y =

  32
  50
```

# そのほかの行列演算

## 要素和

```
A = ones(3, 3)
B = ones(3, 3)
C = A + B
```

```
C =

     2     2     2
     2     2     2
     2     2     2
```

## 要素積

```
A = ones(3, 3)
B = ones(3, 3)
C = A.*B
```

```
C =

     1     1     1
     1     1     1
     1     1     1
```

## 要素べき乗

```
A = [2,2,2; 2,2,2]
B = [1,2,3; 4,5,6]
C = A.^B
```

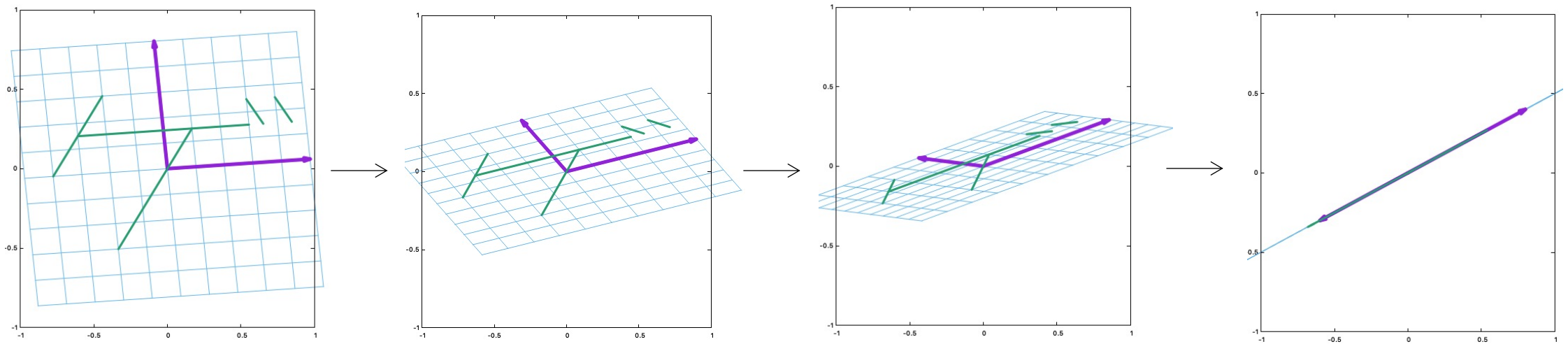
```
C =

     2     4     8
    16    32    64
```

※ 要素和および要素差は「A.+B」「A.-B」でもOK

# 線形代数の基礎：逆行列

- **逆行列＝逆写像**
- 変換後の$\boldsymbol{y}$から変換元の$\boldsymbol{x}$を求める（$\boldsymbol{x} = A^{-1}\boldsymbol{y}$）
- 先ほどの例のように，空間が潰れる場合は異なるベクトル$\boldsymbol{x}, \boldsymbol{x}'$が同じ$\boldsymbol{y}$に移るため，$\boldsymbol{y}$から$\boldsymbol{x}, \boldsymbol{x}'$を推定できない＝逆行列が存在しない

例）$A = \begin{bmatrix} 0.8 & -0.6 \\ 0.4 & -0.3 \end{bmatrix}$ による空間全体の変化（次元が潰れる＝変換元がわからない＝逆行列が求まらない）



# 逆行列 w/ Octave

逆行列に関する演算はOctaveでは2種類存在（$\boldsymbol{x} = A^{-1}\boldsymbol{y}$を求める）

- inv($A$)
- $A \setminus \boldsymbol{y}$
- 後者の方が高速であるため，基本的には「$A \setminus \boldsymbol{y}$」利用がベター

```
A = [1,2; 2,3]
x = [1;-1]

y = A*x
x1 = inv(A) * y
x2 = A\y
```

```
A =

   1   2
   2   3

x =

   1
  -1

x1 =

   1
  -1

x2 =

   1
  -1
```

# 余談

- 連立1次方程式$\boldsymbol{y} = A\boldsymbol{x}$を解く場面は頻出するため，$A^{-1}$を使用する機会も多い
  - 微分方程式，固有方程式, …
- しかし，**逆行列の算出コストは高い**ため，実際は素直に逆行列を算出せずに連立1次方程式を解くことがほとんど

**Q.** ではどうしているのか？

**A. 行列$A$を逆行列の計算が簡単な行列の積に分解してから計算**（Octave内部でも同様）

- LU分解（下三角行列と上三角行列の積に分解）
- QR分解（実直交行列と上三角行列の積に分解）など

LU分解の場合

$A = LU$と分解できると，$x = U^{-1}(L^{-1}\boldsymbol{y})$ となり，$L\boldsymbol{z} = \boldsymbol{y}$，$U\boldsymbol{x} = \boldsymbol{z}$ を順番に解けばよい．
そして，$L, U$が三角行列の場合はこれらの計算が簡単にできる

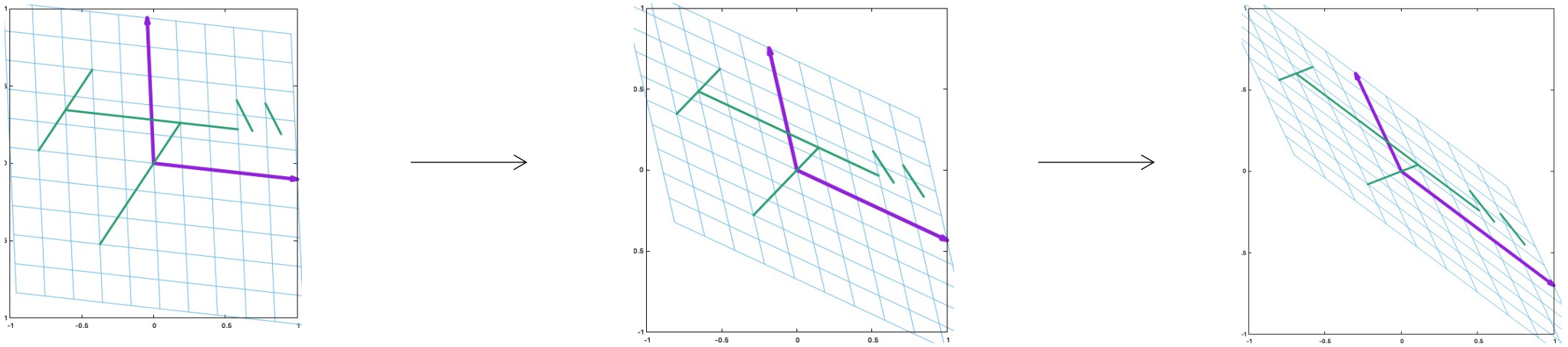
# 線形代数の基礎：固有値・固有値ベクトル

**固有値・固有ベクトル**は行列の重要な情報を表す

例）**変換先の方向を示す**

$A = \begin{bmatrix} 1 & -0.3 \\ -0.7 & 0.6 \end{bmatrix}$ の固有値 $\binom{1.3}{0.3}$ に対応する固有ベクトルはそれぞれ $\binom{0.7071}{-0.7071}$, $\binom{0.3939}{0.9191}$

行列Aを繰り返しかけていくと絶対値が最大の固有値に対応する固有ベクトルの方向に近づいていく



# 線形代数の基礎：固有値・固有値ベクトル

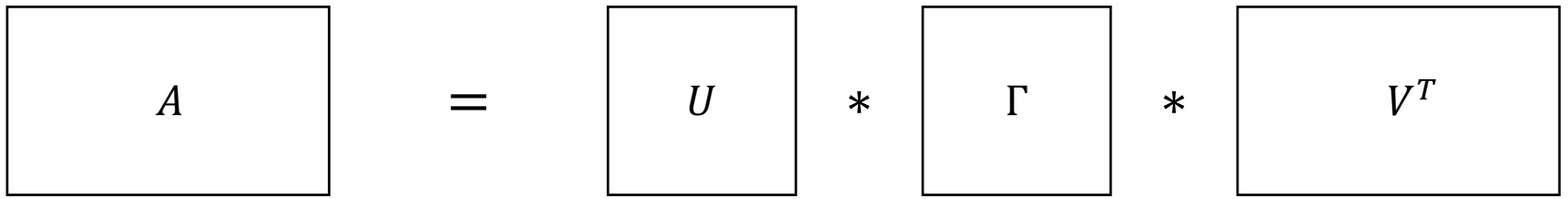
**固有値・固有ベクトル**は行列の重要な情報を表す

例）**特異値分解**

任意の行列$A$は次の形に分解できる（$U, V$：直交行列，$\Gamma$：対角行列）

$$A = U\Gamma V^T$$

$U, V$はそれぞれ行列$AA^T, A^TA$の固有ベクトル，$\Gamma^2 = (AA^T, A^TA$の固有値）



# 線形代数の基礎：固有値・固有値ベクトル

例）**特異値分解**

$$\boxed{\Gamma} = \boxed{\begin{matrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \lambda_3 \end{matrix}}$$

$A \in R^{3\times3}$ かつ $\lambda_1 > \lambda_2 > \lambda_3$ だとすると，

$A = U\Gamma T^T$

$$= \begin{pmatrix} \lambda_1 u_{11} v_{11} & \lambda_1 u_{11} v_{12} & \lambda_1 u_{11} v_{13} \\ \lambda_1 u_{21} v_{11} & \lambda_1 u_{21} v_{12} & \lambda_1 u_{21} v_{13} \\ \lambda_1 u_{31} v_{11} & \lambda_1 u_{31} v_{12} & \lambda_1 u_{31} v_{13} \end{pmatrix} + \begin{pmatrix} \lambda_2 u_{12} v_{21} & \lambda_2 u_{12} v_{22} & \lambda_2 u_{12} v_{23} \\ \lambda_2 u_{22} v_{21} & \lambda_2 u_{22} v_{22} & \lambda_2 u_{22} v_{23} \\ \lambda_2 u_{32} v_{21} & \lambda_2 u_{32} v_{22} & \lambda_2 u_{32} v_{23} \end{pmatrix} + \begin{pmatrix} \lambda_3 u_{13} v_{31} & \lambda_3 u_{13} v_{32} & \lambda_3 u_{13} v_{33} \\ \lambda_3 u_{23} v_{31} & \lambda_3 u_{22} v_{32} & \lambda_3 u_{23} v_{33} \\ \lambda_3 u_{33} v_{31} & \lambda_3 u_{32} v_{32} & \lambda_3 u_{33} v_{33} \end{pmatrix}$$

$$\boxed{A} = \boxed{U} * \boxed{\Gamma} * \boxed{V^T}$$

$$= \boxed{\lambda_1\text{に関する行列}} + \boxed{\lambda_2\text{に関する行列}} + \boxed{\lambda_3\text{に関する行列}}$$

# 線形代数の基礎：固有値・固有値ベクトル

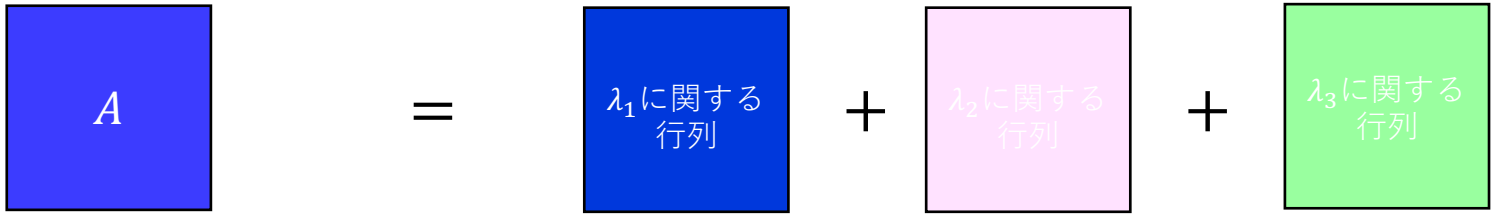
例）**特異値分解**

$$A = U\Gamma T^T$$

$$= \begin{pmatrix} \lambda_1 u_{11} v_{11} & \lambda_1 u_{11} v_{12} & \lambda_1 u_{11} v_{13} \\ \lambda_1 u_{21} v_{11} & \lambda_1 u_{21} v_{12} & \lambda_1 u_{21} v_{13} \\ \lambda_1 u_{31} v_{11} & \lambda_1 u_{31} v_{12} & \lambda_1 u_{31} v_{13} \end{pmatrix} + \begin{pmatrix} \lambda_2 u_{12} v_{21} & \lambda_2 u_{12} v_{22} & \lambda_2 u_{12} v_{23} \\ \lambda_2 u_{22} v_{21} & \lambda_2 u_{22} v_{22} & \lambda_2 u_{22} v_{23} \\ \lambda_2 u_{32} v_{21} & \lambda_2 u_{32} v_{22} & \lambda_2 u_{32} v_{23} \end{pmatrix} + \begin{pmatrix} \lambda_3 u_{13} v_{31} & \lambda_3 u_{13} v_{32} & \lambda_3 u_{13} v_{33} \\ \lambda_3 u_{23} v_{31} & \lambda_3 u_{22} v_{32} & \lambda_3 u_{23} v_{33} \\ \lambda_3 u_{33} v_{31} & \lambda_3 u_{32} v_{32} & \lambda_3 u_{33} v_{33} \end{pmatrix}$$

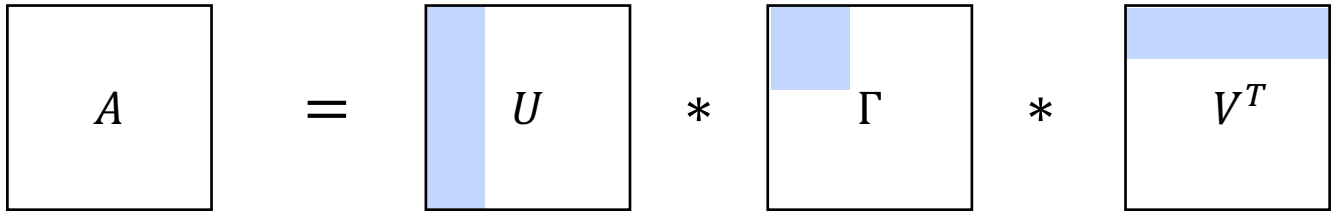
$\lambda_1 \gg \lambda_2 > \lambda_3$ だとすると，$\lambda_2$や$\lambda_3$に関する行列をなくしても行列$A$を十分表現できるはず

＝ **行列の低ランク近似**



# 画像の低ランク近似例

- $A \in R^{256\times256}$の画像に対して，低ランク近似を適用
- 上位50個の特異値および特異ベクトルだけでも十分にもとの画像を表現可能なことがわかる




原画像

上位50個の$\lambda$で再構築

上位5個の$\lambda$で再構築

# 固有値・固有値ベクトル w/ Octave

## 固有値および固有ベクトルの算出方法

- [V, D] = eig(A)
- Vの各列に固有ベクトル，Dの対角成分に固有値

```
## 行列Aの固有値ベクトルと固有値を算出
A = [1.0, -0.3; -0.7, 0.6]
[V, D] = eig(A)
```

```
A =

   1.0000  -0.3000
  -0.7000   0.6000

V =

   0.7071   0.3939
  -0.7071   0.9191

D =

Diagonal Matrix

   1.3000         0
        0   0.3000
```

# 特異値分解 w/ Octave

## 特異値分解の方法

- [U, S, V] = svd(A)
- Uの各列に左特異ベクトル，Sの対角成分に特異値，Vの各列に右特異ベクトル

```
## SVD
A = [1,2,3; 4,5,6; 7,8,9]
[U, S, V] = svd(A)
```

```
A =

   1   2   3
   4   5   6
   7   8   9

U =

  -0.2148   0.8872   0.4082
  -0.5206   0.2496  -0.8165
  -0.8263  -0.3879   0.4082

S =

Diagonal Matrix

   1.6848e+01            0            0
            0   1.0684e+00            0
            0            0   1.4728e-16

V =

  -0.479671  -0.776691   0.408248
  -0.572368  -0.075686  -0.816497
  -0.665064   0.625318   0.408248
```

# 演習課題

**画像（lena_gray.jpg）を読み込み，特異値分解＋低ランク近似＋近似後の画像表示を行うプログラムを書いてください**

- 特異値上位20個および上位100個の特異値と特異ベクトルによるそれぞれの再構築結果を提出（コードも）
- 画像はgoogle classroom上からダウンロード（第3回講義資料のところ）
- imshow関数で画像を表示するときには，**imshow(uint8(img))** のように行列imgをuint8に変換してから実行する

# 演習課題：ヒント

## 特定の行および列を行列から取り出す方法

```
## 行列Aの1～2行目を抽出
A = [1,2,3; 4,5,6; 7,8,9]
B = A(1:2, :)
```

```
A =

   1   2   3
   4   5   6
   7   8   9

B =

   1   2   3
   4   5   6
```

```
## 行列Aの2～3列目を抽出
A = [1,2,3; 4,5,6; 7,8,9]
B = A(:, 2:3)
```

```
A =

   1   2   3
   4   5   6
   7   8   9

B =

   2   3
   5   6
   8   9
```