演習：主成分分析 NumPy

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.decomposition import PCA
plt.style.use('ggplot')
```

訓練データ生成

In [2]:

```
def gen_data(n_samples):
    """多変量正規分布に従う乱数を生成する"""
    mean = [0, 0]  # 平均
    cov = [[2, 0.7], [0.7, 1]]  # 共分散行列を作成
    X = np.random.multivariate_normal(mean, cov, n_samples)
    return X
```

In [3]:

```
n_samples = 100
```

In [4]:

```
X = gen_data(n_samples)  # 訓練データ生成
```

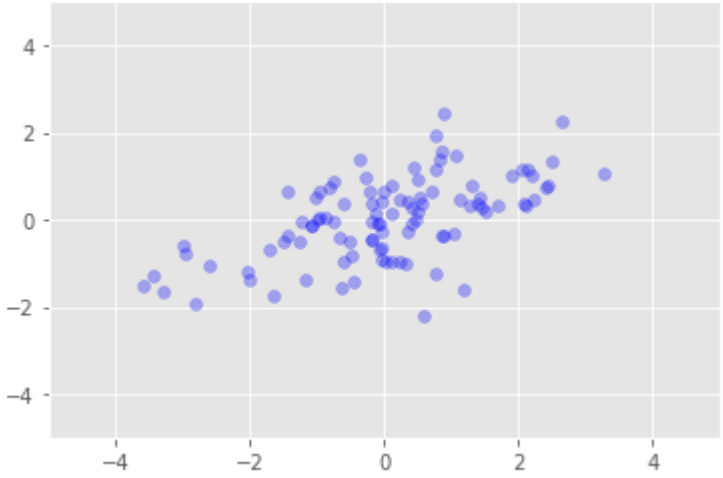
In [5]:

```
print(X.shape)
print(X[:3])
```

```
(100, 2)
[[ 0.54151787  0.49558353]
 [-0.08992738 -0.09450669]
 [-1.16120719 -1.36110861]]
```

In [32]:

```
# 多変量データの散布図を描画
plt.scatter(X[:, 0], X[:, 1], color='b', alpha=0.3)
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.show()
```



学習

訓練データ $X = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$ に対して $\mathbb{E}[x] = 0$ となるように変換する。
すると、不偏共分散行列は $Var[x] = \frac{1}{n-1} X^{\mathrm{T}} X$ と書ける。
$Var[x]$ を固有値分解し、固有値の大きい順に対応する固有ベクトルを第1主成分(w_1), 第2主成分(w_2), ...とよぶ。

In [7]:

```
def get_moments(X):
    mean = X.mean(axis=0)  # 多変量データの平均
    X = X - mean
    standard_cov = np.dot(X.T, X) / (len(X) - 1)  # 不偏共分散行列を求める
    return mean, standard_cov
```

In [8]:

```
def get_components(eig_vectors, n_components):
    """ 固有ベクトルから主成分を取得する

    Parameter
    ----------
    eig_vectors : np.ndarray
        固有ベクトル
    n_components : int
        主成分の数
    """
    W = eig_vectors[:, ::-1][:, :n_components]
    return W.T
```

In [9]:

```
n_components = 2
```

In [10]:

```
#分散共分散行列を標準化
mean, standard_cov = get_moments(X)
```

In [11]:

```
print(mean)
print(standard_cov)
```

```
[0.03126535 0.02688532]
[[2.01469386 0.77695508]
 [0.77695508 0.88676454]]
```

In [12]:

```
# 固有値と固有ベクトルを計算
# np.linalg.eigh はエルミート行列を入力として固有値と固有ベクトルを出力する
eig_values, eig_vectors = np.linalg.eigh(standard_cov)
```

In [13]:

```
print(eig_values)  # 固有値
print(eig_vectors)  # 固有ベクトル
```

```
[0.49066913 2.41078927]
[[ 0.45418805 -0.89090584]
 [-0.89090584 -0.45418805]]
```

In [15]:

```
components = get_components(eig_vectors, n_components)
```

In [16]:

```
print(components)
```

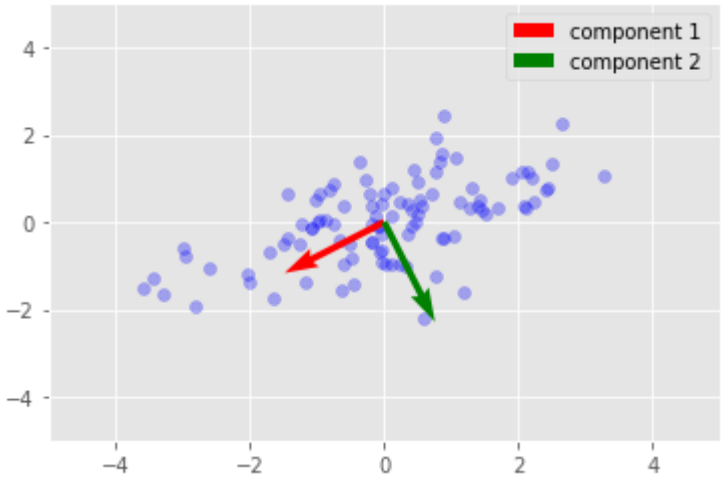
```
[[-0.89090584 -0.45418805]
 [ 0.45418805 -0.89090584]]
```

In [17]:

```
w1 = components[0, :]
w2 = components[1, :]
```

In [18]:

```
# 元データの散布図
plt.scatter(X[:, 0], X[:, 1], color='b', alpha=0.3)
# 第1主成分： (0, 0)を中心に、w1[0]からw1[1]へのベクトルを描画
plt.quiver(0, 0, w1[0], w1[1], width=0.01, scale=6, color='red', label='component 1')
# 第2主成分： (0, 0)を中心に、w2[0]からw2[1]へのベクトルを描画
plt.quiver(0, 0, w2[0], w2[1], width=0.01, scale=6, color='green', label='component 2')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.legend()
plt.show()
```



考察：
最も散らばっている方向にcomponent1のベクトルが向いている。
また、それと直行するようにcomponent2のベクトルが向いている。
つまり、主成分分析はうまくいっているように見える。

変換（射影）

元のデータを m 次元に射影するときは、行列 W を $W = [w_1, w_2, \cdots, w_m]$ とし、データ点 x を $z = W^{\mathrm{T}}x$ によって変換(射影)する。
よって、データ X に対しては $Z = X^{\mathrm{T}}W$ によって変換する。

In [37]:

```
def transform_by_pca(X, pca):
    """データXをpcaによって変換する"""
    mean = X.mean(axis=0)
    X = X - mean
    Z = np.dot(X, pca)
    return Z
```

In [38]:

```
Z = transform_by_pca(X, components.T)
```

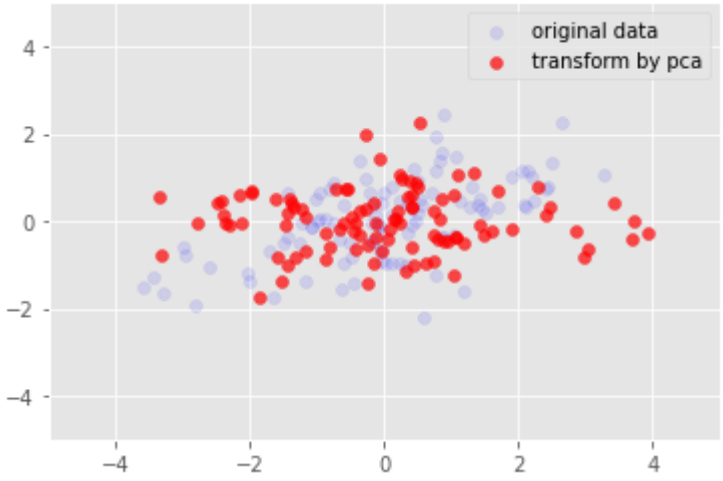
In [39]:

```
print(Z.shape)
```

```
(100, 2)
```

In [40]:

```
# 元データの散布図
plt.scatter(X[:, 0], X[:, 1], color='b', alpha=0.1, label='original data')
# 変換後の散布図
plt.scatter(Z[:, 0], Z[:, 1], color='r', alpha=0.7, label='transform by pca')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.legend()
plt.show()
```



逆変換

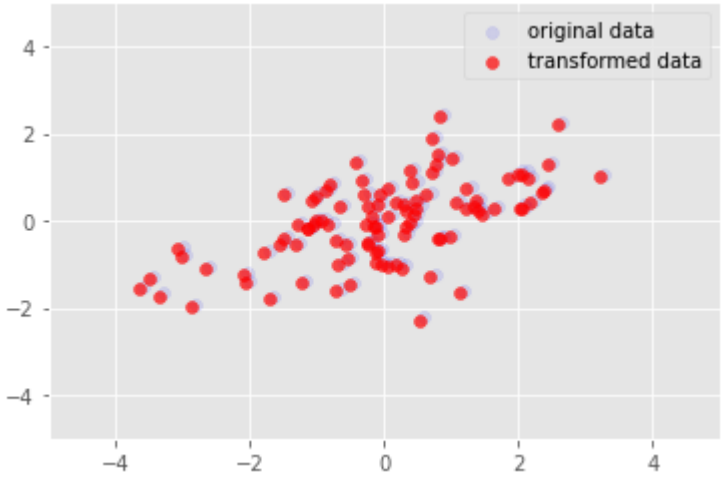
射影されたデータ点 z を元のデータ空間へ逆変換するときは $\bar{x} = (W^{\mathrm{T}})^{-1}z = Wz$ によって変換する。
よって、射影されたデータ Z に対しては $\bar{X} = ZW^{\mathrm{T}}$ によって変換する。

In [47]:

```
mean = X.mean(axis=0)
X_ = np.dot(Z, components) - mean
```

In [48]:

```
# 元データの散布図
plt.scatter(X[:, 0], X[:, 1], color='b', alpha=0.1, label='original data')
plt.scatter(X_[:, 0], X_[:, 1], color='r', alpha=0.7, label='transformed data')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.legend()
plt.show()
```



ほぼ、元のデータ空間に戻っていると言える。

（比較用）sklearnで主成分分析

In [28]:

```
pca = PCA(n_components=2)
pca.fit(X)
```

Out[28]:

PCA(n_components=2)

In [29]:

```
print(f'主成分:\n{pca.components_}')
print(f'平均: {pca.mean_}')
print(f'共分散:\n{pca.get_covariance()}')
```

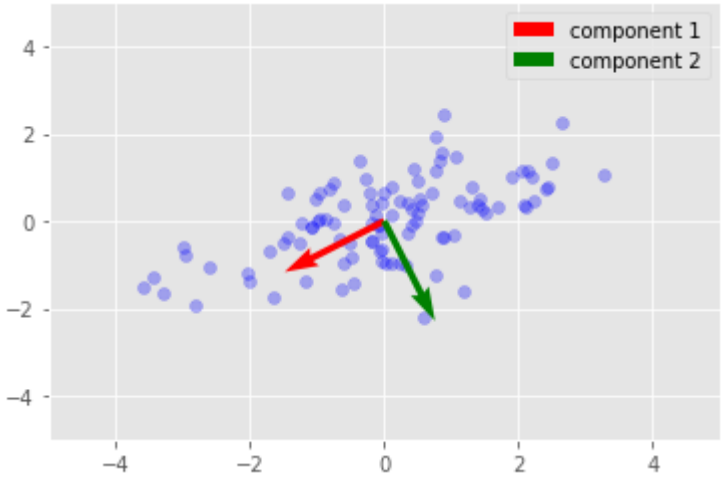
主成分:
[[-0.89090584 -0.45418805]
 [0.45418805 -0.89090584]]
平均: [0.03126535 0.02688532]
共分散:
[[2.01469386 0.77695508]
 [0.77695508 0.88676454]]

In [30]:

```
w1 = pca.components_[0, :]
w2 = pca.components_[1, :]
```

In [31]:

```
# 元データの散布図
plt.scatter(X[:, 0], X[:, 1], color='b', alpha=0.3)
# 第1主成分： (0, 0)を中心に、w1[0]からw1[1]へのベクトルを描画
plt.quiver(0, 0, w1[0], w1[1], width=0.01, scale=6, color='red', label='component 1')
# 第2主成分： (0, 0)を中心に、w2[0]からw2[1]へのベクトルを描画
plt.quiver(0, 0, w2[0], w2[1], width=0.01, scale=6, color='green', label='component 2')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.legend()
plt.show()
```



NumPyによる実装と同じ結果となった。

In [ ]:

```
```