|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н. Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н. Э. Баумана)** |

# Анализ независимых компонент

Студенты группы ИУ10-99

Шумаков А. Ю.

Замотаев И. Ю.

Гаврилов Ю.Р.

Смольянинов А. А.

**Москва, 2022**

Оглавление

[Постановка задачи 3](#_Toc117191369)

[Статья с примером решения задачи 3](#_Toc117191370)

[Данные для анализа 3](#_Toc117191371)

[Анализ независимых компонент 4](#_Toc117191372)

[Алгоритм анализа независимых компонент 5](#_Toc117191373)

[Теоретическое обоснование метода решения 6](#_Toc117191374)

[Реализация на Python 9](#_Toc117191375)

[Вывода 15](#_Toc117191376)

[Приложение 16](#_Toc117191377)

# Постановка задачи

## Статья с примером решения задачи

https://towardsdatascience.com/independent-component-analysis-ica-in-python-a0ef0db0955e

## Данные для анализа

https://disk.yandex.ru/d/UDMccnU5zvk\_IQ

https://disk.yandex.ru/d/UcHAWNjzxlnj6g

# Анализ независимых компонент

АНК — это вычислительный метод, разделяющий многомерный сигнал на компоненты. Используя АНК, можно выделить нужный компонент из комплекса сигналов.

ICA можно объяснить на примере вечеринки. Представим, что в комнате есть несколько микрофонов и несколько людей, которые разговаривают, поют, шумят. Задача – разделить пение, разговоры и шум. Для этого и применяется метод независимых компонент.

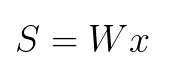
Метод помогает выделять IC (независимые компоненты, в данном случае разговор и музыку) из смеси звуков, сигналов. После применения ICA получают разделенные сигналы.

Все сигналы смешаны и взаимно искажают характеристики, накладываясь друг на друга. Однако стоит заметить, что, несмотря на наложение, сигналы друг от друга независимы – это одно из ключевых положений метода независимых компонент. Соответственно, из совокупности полученных сигналов можно выделить необходимую составляющую. Для этого параметры сигнала (его амплитуда, частота, спектр) записываются в виде математических величин в специальную таблицу, элементы которой взаимосвязаны. Затем, с помощью ряда математических операций, из этой таблицы вычленяются элементы, относящиеся к тем или иным компонентам. Т.е. происходит разделение всей совокупности величин на группы, каждая из которых описывает тот или иной элемент итогового сигнала.

Таким образом, после разделения записанного сигнала мы получаем возможность удалить из него ненужные для нас элементы – шум голосов и музыку в примере с вечеринкой или моргание и другие помехи в случае ЭЭГ. Это позволяет нам очистить запись от лишних данных или же выделить конкретную информацию.

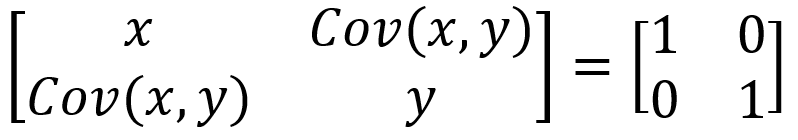
# Алгоритм анализа независимых компонент

1. Центрирование массива **x**вычитанием среднего значения.
2. Избавление от корреляции.
3. Выбор случайного начального значения для демикширования матрицы **w.**
4. Расчет нового значения **w.**
5. Нормализация **w.**
6. Проверка сходимости алгоритма. Если сходимости нет — возврат к шагу 4.
7. Вычисление скалярного произведения **w** и **x** для получения независимых источников сигналов.

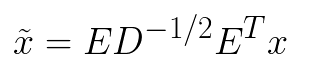


# Теоретическое обоснование метода решения

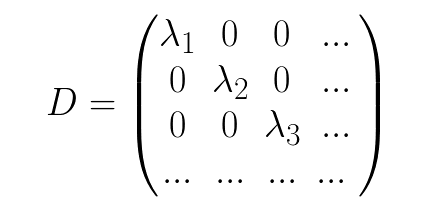
Прежде чем применять алгоритм АНК, необходимо избавиться от корреляции. Это означает, что мы преобразовываем сигнал таким образом, чтобы потенциальные корреляции между его компонентами исчезли (ковариация стала равна 0), а дисперсия каждого компонента стала равна 1. Также можно сказать, что ковариационная матрица будет равна единичной матрице.



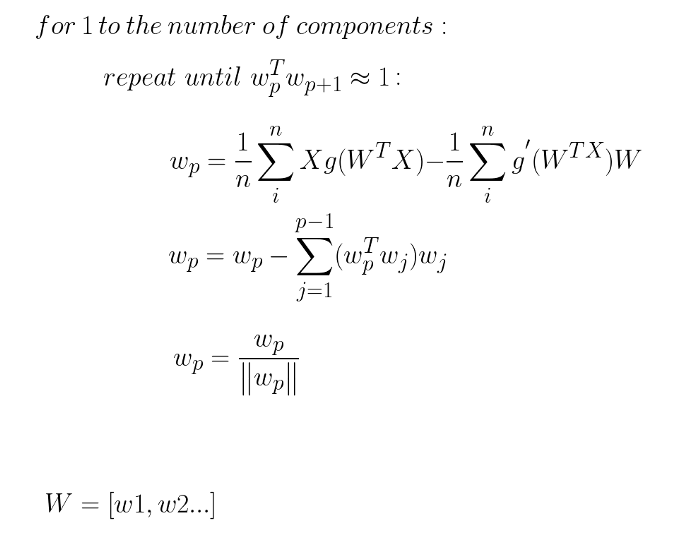
Фактический способ, которым мы избавляемся от корреляции, включает в себя **разложение по собственным значениям его ковариационной матрицы**. Соответствующее математическое уравнение может быть описано так:



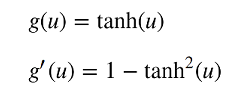
Где **D** — это диагональная матрица собственных значений (каждая лямбда является собственным значением ковариационной матрицы), а **E** — ортогональная матрица собственных значений.



Завершив предварительную обработку сигнала, для каждого компонента обновляем значения демикшированной матрицы **w** до тех пор, пока алгоритм не сойдется или не будет достигнуто максимальное количество итераций. Когда скалярное произведение **w** и ее транспонированной матрицы примерно равно 1, сходимость считается достигнутой.



где



# Реализация на Python

Импортируем необходимые библиотеки:

**import** numpy **as** np

np.random.seed(0)

**from** scipy **import** signal

**from** scipy.io **import** wavfile

**from** matplotlib **import** pyplot **as** plt

**import** seaborn **as** sns

sns.set(rc={'figure.figsize':(11.7,8.27)})

Задаем функции для вычисления **g** и **g’** которые будем использовать для определения нового значения **w**:

**def** g(x):

**return** np.tanh(x)

**def** g\_der(x):

**return** 1 - g(x) \* g(x)

Создаем функцию, чтобы центрировать сигнал вычитанием среднего:

**def** center(X):

X = np.array(X)

mean = X.mean(axis=1, keepdims=True)

**return** X- mean

Определяем функцию для избавления от корреляции:

**def** whitening(X):

cov = np.cov(X)

d, E = np.linalg.eigh(cov)

D = np.diag(d)

D\_inv = np.sqrt(np.linalg.inv(D))

X\_whiten = np.dot(E, np.dot(D\_inv, np.dot(E.T, X)))

**return** X\_whiten

Определяем функцию для обновления демикшированной матрицы **w**:

**def** calculate\_new\_w(w, X):

w\_new = (X \* g(np.dot(w.T, X))).mean(axis=1) - g\_der(np.dot(w.T, X)).mean() \* w

w\_new /= np.sqrt((w\_new \*\* 2).sum())

**return** w\_new

Определяем основной метод, вызывающий функции предварительной обработки, инициализирующий **w**случайным набором значений и итерационно обновляющий **w**. На сходимость указывает тот факт, что идеальная **w** будет ортогональной, следовательно, произведение **w** и ее транспонированной матрицы будет примерно равно 1. После расчета оптимального значения **w** для каждого компонента возьмем скалярное произведение полученной матрицы и сигнала **x**, чтобы получить источники:

**def** ica(X, iterations, tolerance=1***e***-5):

X = center(X)

X = whitening(X)

components\_nr = X.shape[0]

W = np.zeros((components\_nr, components\_nr), dtype=X.dtype)

**for** i **in** range(components\_nr):

w = np.random.rand(components\_nr)

**for** j **in** range(iterations):

w\_new = calculate\_new\_w(w, X)

**if** i >= 1:

w\_new -= np.dot(np.dot(w\_new, W[:i].T), W[:i])

distance = np.abs(np.abs((w \* w\_new).sum()) - 1)

w = w\_new

**if** distance < tolerance:

**break**

W[i, :] = w

S = np.dot(W, X)

**return** S

Определяем функцию для построения и сравнения оригинальных, смешанных и предсказанных сигналов:

**def** plot\_mixture\_sources\_predictions(X, original\_sources, S):

fig = plt.figure()

plt.subplot(3, 1, 1)

**for** x **in** X:

plt.plot(x)

plt.title("mixtures")

plt.subplot(3, 1, 2)

**for** s **in** original\_sources:

plt.plot(s)

plt.title("real sources")

plt.subplot(3,1,3)

**for** s **in** S:

plt.plot(s)

plt.title("predicted sources")

fig.tight\_layout()

plt.show()

Для примера создадим метод для искусственного смешивания сигналов из различных источников:

**def** mix\_sources(mixtures, apply\_noise=False):

**for** i **in** range(len(mixtures)):

max\_val = np.max(mixtures[i])

**if** max\_val > 1 **or** np.min(mixtures[i]) < 1:

mixtures[i] = mixtures[i] / (max\_val / 2) - 0.5

X = np.c\_[[mix **for** mix **in** mixtures]]

**if** apply\_noise:

X += 0.02 \* np.random.normal(size=X.shape)

**return** X

Создадим три сигнала, каждый из которых имеет собственный шаблон:

n\_samples = 2000

time = np.linspace(0, 8, n\_samples)

s1 = np.sin(2 \* time) # синусоидальная волна

s2 = np.sign(np.sin(3 \* time)) # меандр

s3 = signal.sawtooth(2 \* np.pi \* time) # пилообразная волна

Вычисляем скалярное произведение матрицы **A**и сигналов для получения комбинации всех трех. Затем используем АНК, чтобы разделить смешанный сигнал на исходные:

X = np.c\_[s1, s2, s3]

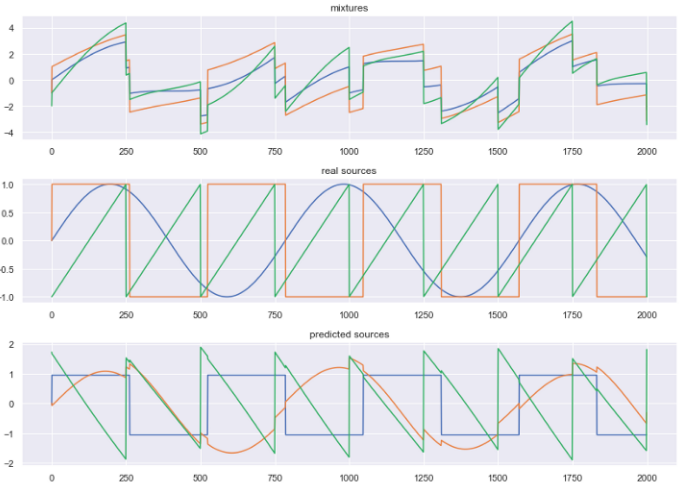
A = np.array(([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]]))

X = np.dot(X, A.T)

X = X.T

S = ica(X, iterations=1000)

plot\_mixture\_sources\_predictions(X, [s1, s2, s3], S)



**График 1. Полученное разложение исходных сигналов**

Используем АНК для разложения смеси реальных аудиодорожек и отображения результата.

sampling\_rate, mix1 = wavfile.read('mix1.wav')

sampling\_rate, mix2 = wavfile.read('mix2.wav')

sampling\_rate, source1 = wavfile.read('source1.wav')

sampling\_rate, source2 = wavfile.read('source2.wav')

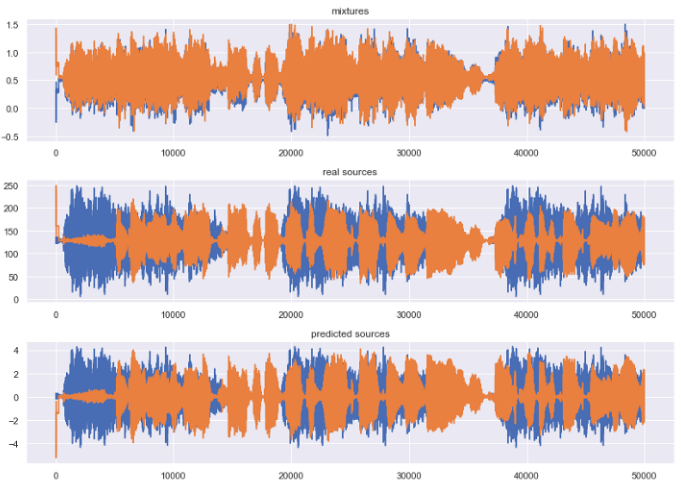
X = mix\_sources([mix1, mix2])

S = ica(X, iterations=1000)

plot\_mixture\_sources\_predictions(X, [source1, source2], S)

wavfile.write('out1.wav', sampling\_rate, S[0])

wavfile.write('out2.wav', sampling\_rate, S[1])



**График 2. Полученное разложение смеси реальных аудиодорожек**

Для примера воспользуемся **scikit-learn**, чтобы сравнить результаты.

**from** sklearn.decomposition **import** FastICA

np.random.seed(0)

n\_samples = 2000

time = np.linspace(0, 8, n\_samples)

s1 = np.sin(2 \* time)

s2 = np.sign(np.sin(3 \* time))

s3 = signal.sawtooth(2 \* np.pi \* time)

S = np.c\_[s1, s2, s3]

S += 0.2 \* np.random.normal(size=S.shape)

S /= S.std(axis=0)

A = np.array([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]])

X = np.dot(S, A.T)

ica = FastICA(n\_components=3)

S\_ = ica.fit\_transform(X)

fig = plt.figure()

models = [X, S, S\_]

names = ['mixtures', 'real sources', 'predicted sources']

colors = ['red', 'blue', 'orange']

**for** i, (name, model) **in** enumerate(zip(names, models)):

plt.subplot(4, 1, i+1)

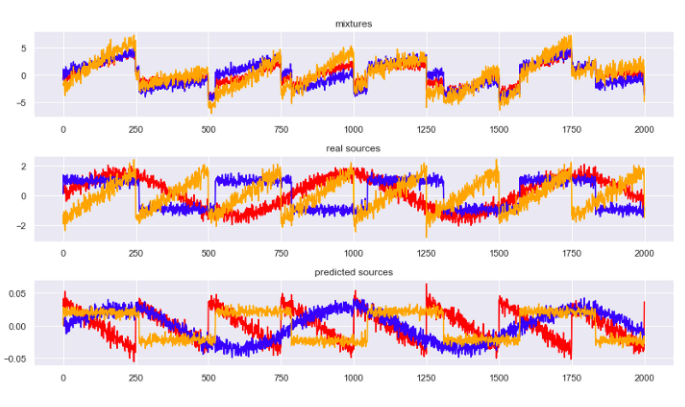
plt.title(name)

**for** sig, color **in** zip (model.T, colors):

plt.plot(sig, color=color)

fig.tight\_layout()

plt.show()



**График 3. Полученное разложение смеси реальных аудиодорожек, полученное с помощью scikit-learn**

# Вывода

Мы рассмотрели, как проблема распознавания и выделения отдельного источника звука на фоне нескольких решается может быть решена с помощью анализа независимых компонент. Мы начали с выделения трех синтетических, искусственно смешанных сигналов, имевших синусоидальный, пилообразный виды и форму меандра. В итоге нам удалось для разложить смесь реальных аудиодорожек и выделить результат, при этом удалось воспользоваться scikit-learn для сравнения с альтернативным результатом.

# Приложение

**import** numpy **as** np

np.random.seed(0)

**from** scipy **import** signal

**from** scipy.io **import** wavfile

**from** matplotlib **import** pyplot **as** plt

**import** seaborn **as** sns

sns.set(rc={'figure.figsize':(11.7,8.27)})

**def** g(x):

**return** np.tanh(x)

**def** g\_der(x):

**return** 1 - g(x) \* g(x)

**def** center(X):

X = np.array(X)

mean = X.mean(axis=1, keepdims=True)

**return** X- mean

**def** whitening(X):

cov = np.cov(X)

d, E = np.linalg.eigh(cov)

D = np.diag(d)

D\_inv = np.sqrt(np.linalg.inv(D))

X\_whiten = np.dot(E, np.dot(D\_inv, np.dot(E.T, X)))

**return** X\_whiten

**def** calculate\_new\_w(w, X):

w\_new = (X \* g(np.dot(w.T, X))).mean(axis=1) - g\_der(np.dot(w.T, X)).mean() \* w

w\_new /= np.sqrt((w\_new \*\* 2).sum())

**return** w\_new

**def** ica(X, iterations, tolerance=1***e***-5):

X = center(X)

X = whitening(X)

components\_nr = X.shape[0]

W = np.zeros((components\_nr, components\_nr), dtype=X.dtype)

**for** i **in** range(components\_nr):

w = np.random.rand(components\_nr)

**for** j **in** range(iterations):

w\_new = calculate\_new\_w(w, X)

**if** i >= 1:

w\_new -= np.dot(np.dot(w\_new, W[:i].T), W[:i])

distance = np.abs(np.abs((w \* w\_new).sum()) - 1)

w = w\_new

**if** distance < tolerance:

**break**

W[i, :] = w

S = np.dot(W, X)

**return** S

**def** plot\_mixture\_sources\_predictions(X, original\_sources, S):

fig = plt.figure()

plt.subplot(3, 1, 1)

**for** x **in** X:

plt.plot(x)

plt.title("mixtures")

plt.subplot(3, 1, 2)

**for** s **in** original\_sources:

plt.plot(s)

plt.title("real sources")

plt.subplot(3,1,3)

**for** s **in** S:

plt.plot(s)

plt.title("predicted sources")

fig.tight\_layout()

plt.show()

**def** mix\_sources(mixtures, apply\_noise=False):

**for** i **in** range(len(mixtures)):

max\_val = np.max(mixtures[i])

**if** max\_val > 1 **or** np.min(mixtures[i]) < 1:

mixtures[i] = mixtures[i] / (max\_val / 2) - 0.5

X = np.c\_[[mix **for** mix **in** mixtures]]

**if** apply\_noise:

X += 0.02 \* np.random.normal(size=X.shape)

**return** X

n\_samples = 2000

time = np.linspace(0, 8, n\_samples)

s1 = np.sin(2 \* time) # синусоидальная волна

s2 = np.sign(np.sin(3 \* time)) # меандр

s3 = signal.sawtooth(2 \* np.pi \* time) # пилообразная волна

X = np.c\_[s1, s2, s3]

A = np.array(([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]]))

X = np.dot(X, A.T)

X = X.T

S = ica(X, iterations=1000)

plot\_mixture\_sources\_predictions(X, [s1, s2, s3], S)

sampling\_rate, mix1 = wavfile.read('mix1.wav')

sampling\_rate, mix2 = wavfile.read('mix2.wav')

sampling\_rate, source1 = wavfile.read('source1.wav')

sampling\_rate, source2 = wavfile.read('source2.wav')

X = mix\_sources([mix1, mix2])

S = ica(X, iterations=1000)

plot\_mixture\_sources\_predictions(X, [source1, source2], S)

wavfile.write('out1.wav', sampling\_rate, S[0])

wavfile.write('out2.wav', sampling\_rate, S[1])

**from** sklearn.decomposition **import** FastICA

np.random.seed(0)

n\_samples = 2000

time = np.linspace(0, 8, n\_samples)

s1 = np.sin(2 \* time)

s2 = np.sign(np.sin(3 \* time))

s3 = signal.sawtooth(2 \* np.pi \* time)

S = np.c\_[s1, s2, s3]

S += 0.2 \* np.random.normal(size=S.shape)

S /= S.std(axis=0)

A = np.array([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]])

X = np.dot(S, A.T)

ica = FastICA(n\_components=3)

S\_ = ica.fit\_transform(X)

fig = plt.figure()

models = [X, S, S\_]

names = ['mixtures', 'real sources', 'predicted sources']

colors = ['red', 'blue', 'orange']

**for** i, (name, model) **in** enumerate(zip(names, models)):

plt.subplot(4, 1, i+1)

plt.title(name)

**for** sig, color **in** zip (model.T, colors):

plt.plot(sig, color=color)

fig.tight\_layout()

plt.show()