2. ВТОРОЙ ЭТАП

Задачи второго этапа

3.1. Блок заданий 1

Справочные материалы к данному блоку задач представлены в Приложении 1. В скобках возле номера задачи указан максимальный балл за данную задачу.

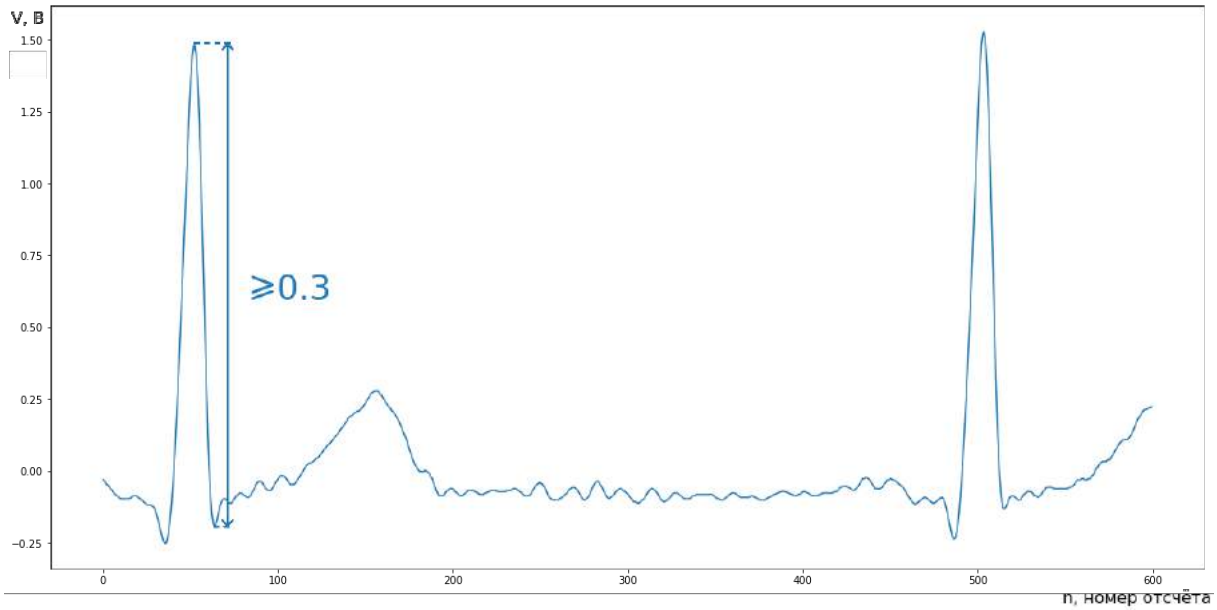
Задача 3.1.1. (8 баллов)

В данной задаче вам предлагается определить количество сокращений сердца на записи сигнала электрокардиограммы (ЭКГ) — электрического сигнала, возникающего при работе сердца человека. Данные, представленные в этой задаче, взяты из архива PhysioBank.

Сигнал на входе представляет собой последовательность действительных чисел, разделенных пробелом и лежащих в диапазоне от -0.5 до 2.0. Сигнал был оцифрован с частотой 500 Гц. Частота сердечных сокращений в тестовых данных не превышает 200 ударов в минуту. Высота R-зубца не меньше 0.3.

Пример входного сигнала с двадцатью тремя сокращениями сердца доступен по ссылке (`https://goo-gl.ru/4Tzv`).

Приведём ниже визуализацию участка примера:



Как вы можете убедиться, на изображенном участке было зарегистрировано два

сокращения сердца. Ваши решения будут тестироваться на более длительном сигнале, для проверки корректности их работы вы можете использовать данный образец.

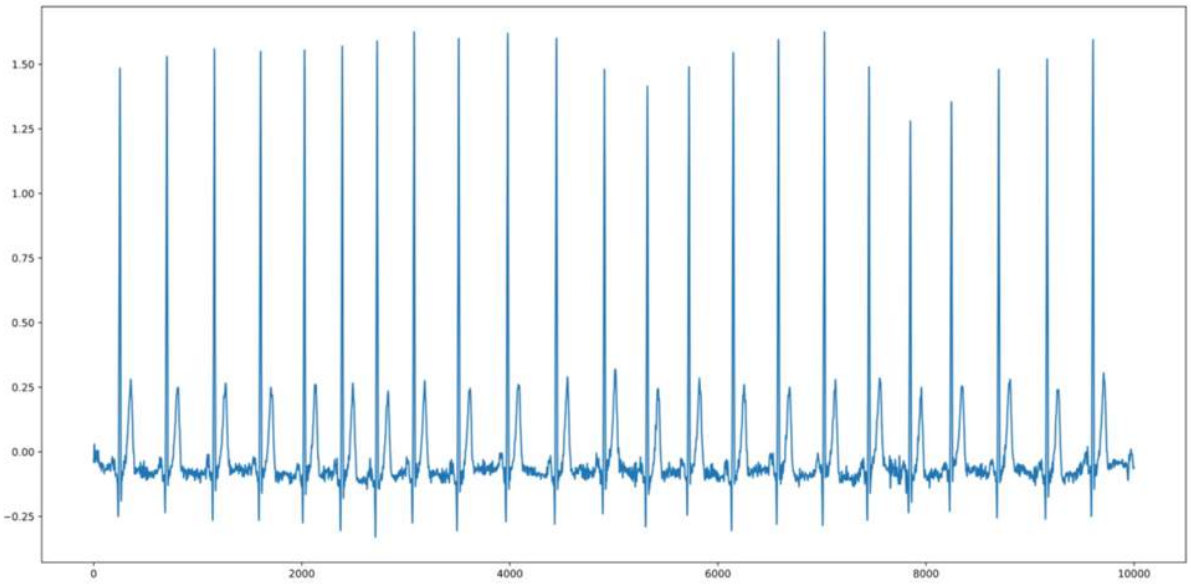
Пример №1

Стандартный ввод
-0.03 -0.035 -0.045 -0.05 -0.06 -0.065 -0.075 -0.08 -0.085 -0.09 -0.095 -0.095 -0.095 -0.095 -0.095 -0.095 -0.09 -0.085 ...
Стандартный вывод
2

Решение

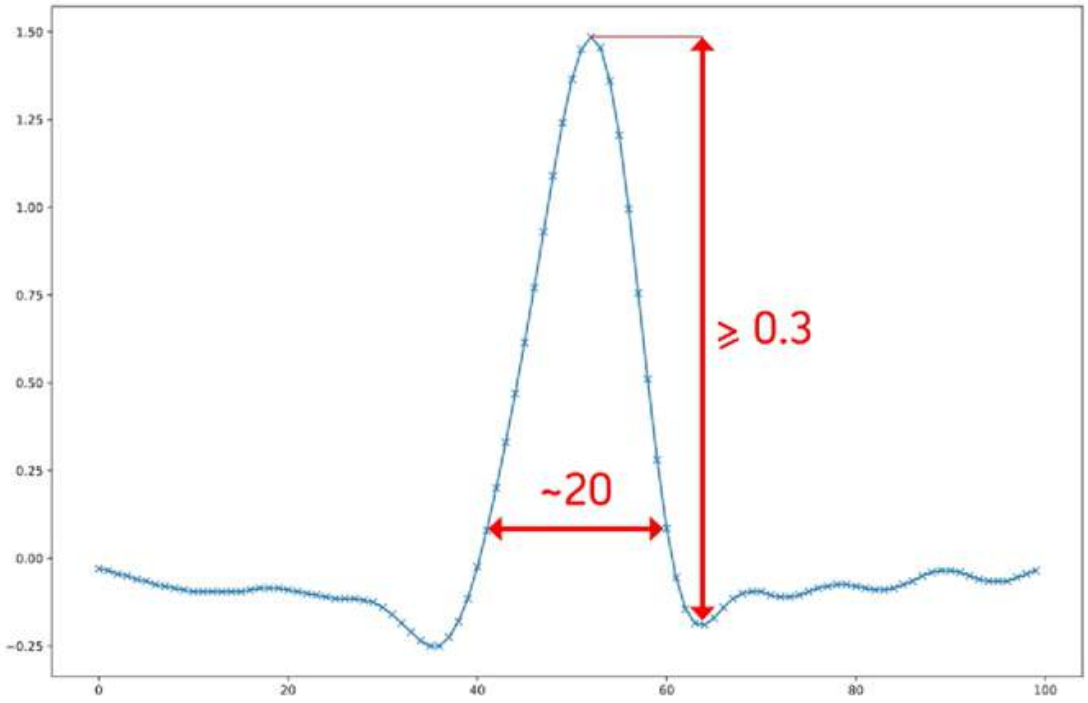
Визуализируем сигнал из данной задачи.

На кардиограмме чётко видны R-зубцы, число которых соответствует сокращениям сердца.

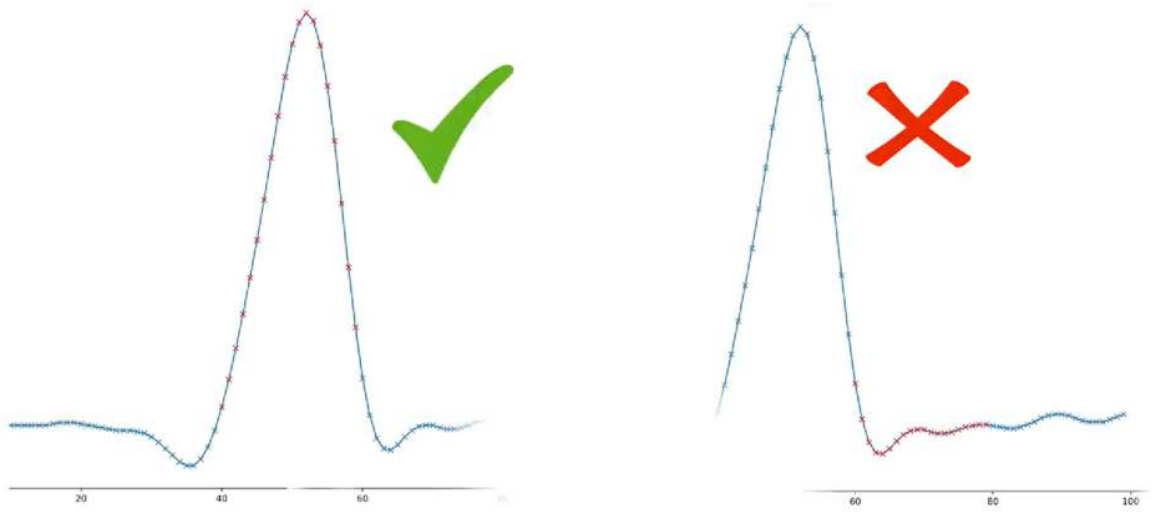


Рассмотрим фрагмент сигнала, содержащий R-зубец. Из рисунка видно, что ширина зубца приблизительно составляет 20 точек. Согласно условию, его высота не менее 0.3. Для поиска R-зубцов будем последовательно рассматривать участки сигнала, содержащие 20 точек. Если середина участка больше каждого из его концов на 0.3, то отрезок содержит R-зубец.

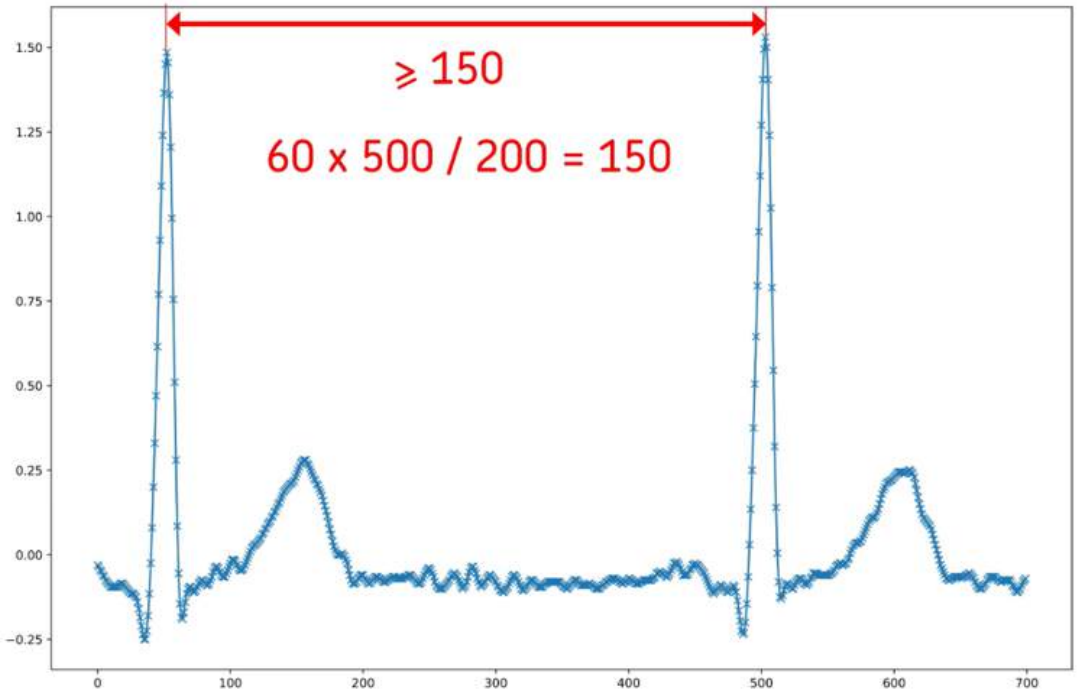
Фрагмент сигнала, содержащий R-зубец.



Фрагменты сигналов, содержащий и не содержащие R-зубец.



По условию задачи сигнал был оцифрован с частотой 500 Гц, а частота сердечных сокращений в тестовых данных не превышает 200 ударов в минуту Следовательно, между вершинами последовательных R-зубцов лежит не менее 150 точек:



С учетом особенностей, выявленных выше возможное решение на языке Python 3 может иметь следующий вид:

```
data = list(map(float, input().split()))
i = 0
beats = 0
while i < len(data) - 20:
    chunk = data[i:i+20]
    if chunk[10]-chunk[0] > 0.3 and chunk[10]-chunk[-1] > 0.3:
        beats += 1
        i += 150
    else:
        i += 1
print(beats)
```

Пояснения к данному решению:

Считаем сигнал в список data.

Введём переменные i и beats. Переменная i - индекс начала участка сигнала, в котором мы будем искать R-зубец, а переменная beats - число обнаруженных R-зубцов.

Значения переменной i будем перебирать в цикле while. Так как переменная i - индекс начала участка сигнала, содержащего 20 точек, то её максимальное значение должно быть на 20 меньше числа значений всего сигнала.

В переменной chunk будем хранить срез сигнала, в котором ищем R-зубец.

Если значение из середины среза больше значений первого и последнего элементов на 0.3, то отрезок содержит R-зубец и мы увеличиваем переменную beats на единицу. При этом чтобы исключить возможность дважды посчитать один и тот

же пик, увеличим индекс начала среза на 150, то есть на минимальное число точек между R-зубцами.

Если условие наличия R-зубца не было удовлетворено, увеличиваем переменную i на 1.

В конце выведем получившееся число зубцов функцией print.

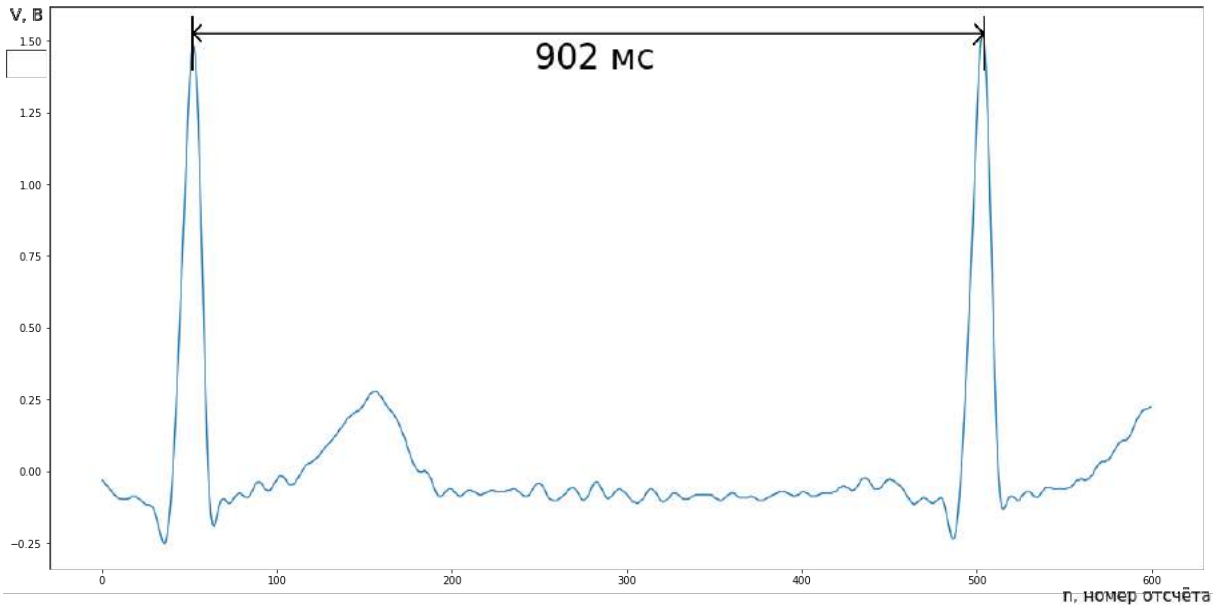
Задача 3.1.2. (10 баллов)

Как и в предыдущей задаче, входными данными в этой задаче являются сигналы ЭКГ из архива PhysioBank. Сигнал на входе представляет собой последовательность действительных чисел, разделенных пробелом и лежащих в диапазоне от -0.5 до 2.0. Сигнал был оцифрован с частотой 500 Гц. Частота сердечных сокращений в тестовых данных не превышает 200 ударов в минуту. Высота R-зубца не меньше 0.3.

Вам предлагается рассчитать временной интервал между вершинами соседних R-зубцов. Если R-зубец имеет на вершине несколько точек с максимальным значением, для расчёта интервала берём первую из этих точек. Ответом является последовательность временных интервалов между вершинами соседних R-зубцов в миллисекундах, разделённых пробелами. Пример входных данных и ответ для данного примера доступен по ссылке:

- пример входных данных (`https://www.dropbox.com/s/5qe3mk9zsb1119h/2.txt?dl=1`)
- ответ для данного примера (`https://www.dropbox.com/s/pu72nkzsf847phz/2_ans.txt?dl=1`)

Для проверки корректности решения используется отрывок из примера с двумя первыми сокращениями сердца:



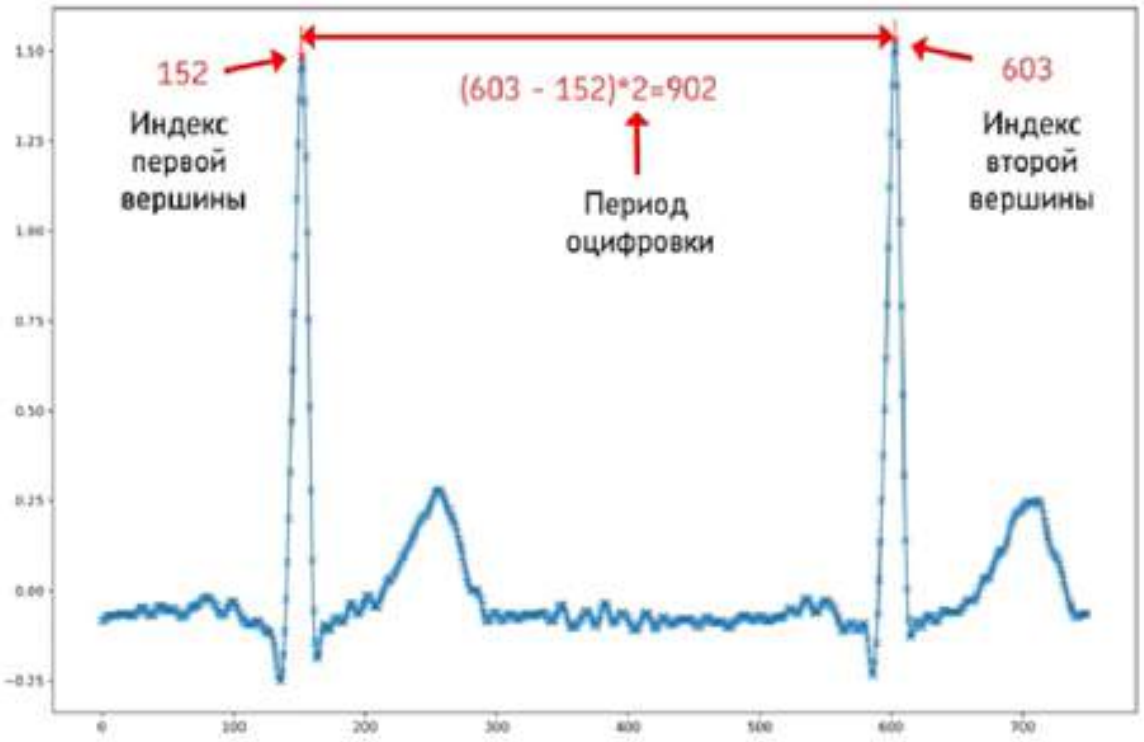
Пример №1

Стандартный ввод
-0.03 -0.035 -0.045 -0.05 -0.06 -0.065 -0.075 -0.08 -0.085 -0.09 -0.095 -0.095 -0.095 -0.095 -0.095 -0.095 -0.09 -0.085 -0.085 ...
Стандартный вывод
902

Решение

R-зубцы будем искать тем же способом, что и в предыдущей задаче. Определив, что срез сигнала содержит зубец, следует найти индекс его вершины.

Зная индексы соседних R-зубцов и период оцифровки сигнала, можно найти временные интервалы между зубцами.



Приступим к написанию программы

Как и в предыдущей задаче, считаем сигнал в список data и введём индекс начала среза сигнала i.

Введём переменную last_index, в которой будет храниться индекс вершины последнего найденного зубца.

Вершину зубца будем искать также, как и в предыдущей задаче.

В переменную current_index запишем индекс вершины нового найденного зубца.

Сравнивая last_index со значением, присвоенным переменной до начала цикла, т.е. нулём, мы проверяем, является ли найденный R-зубец первым. Если это так, то мы присваиваем индекс вершины зубца переменной last_index.

Если же найденный в текущем срезе зубец не первый, мы можем рассчитать временной интервал между ним и предыдущим зубцом, а затем вывести интервал с помощью функции print.

Изменяя переменную i, руководствуемся теми же соображениями что и в предыдущей задаче.

Возможное решение:

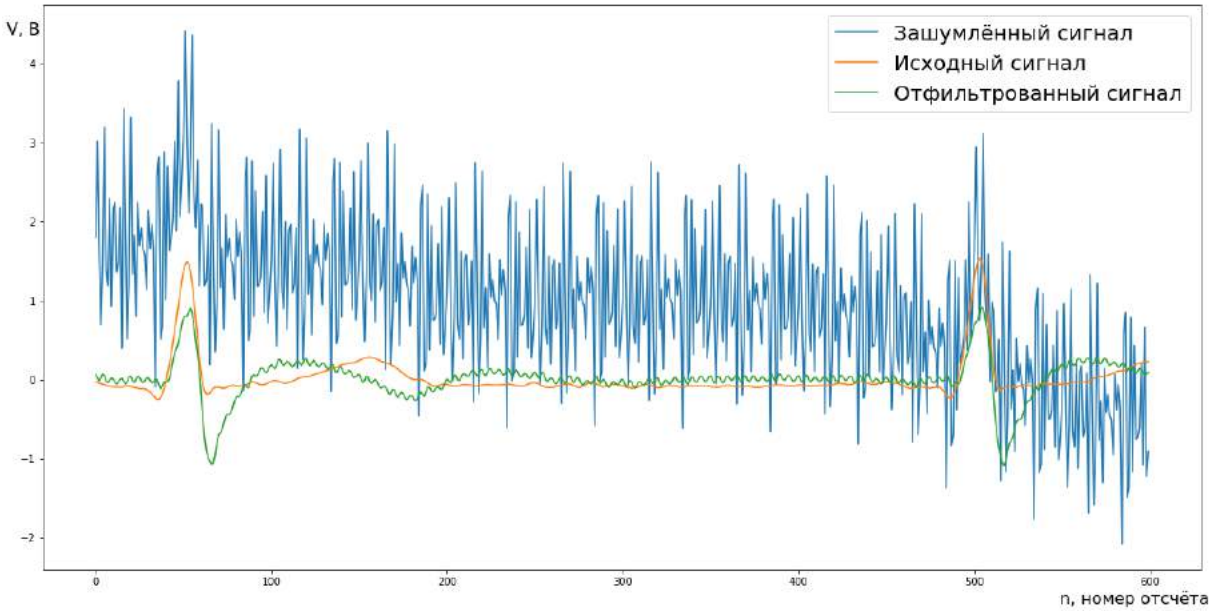
```
data = list(map(float, input().split()))
i = 0
last_index = 0
while i < len(data) - 20:
    chunk = data[i:i+20]
    if chunk[10]-chunk[0] > 0.3 and chunk[10]-chunk[-1] > 0.3:
        current_index = i + chunk.index(max(chunk))
        if last_index == 0:
            last_index = current_index
        else:
            print((current_index - last_index) * 2, end = ' ')
            last_index = current_index
        i += 150
    else:
        i += 1
```

Задача 3.1.3. (12 баллов)

Финальная задача данного блока позволит вам познакомиться с очисткой сигнала от шумов. Мы продолжаем работать с сигналом ЭКГ. В данном примере на сигнал электрокардиограммы был наложен шум. Шум состоит из конечного числа гармонических колебаний заданных частот, одинаковых для всех тестовых данных. Подсчитайте число сердечных сокращений записанных в подаваемом на вход сигнале. Обратите внимание, что после фильтрации вид сигнала изменится, и величина максимумов может уменьшиться.

Пример входного сигнала с двадцатью тремя сокращениями сердца доступен по ссылке (`https://www.dropbox.com/s/ev1k9o0300mzvlz/3.txt?dl=0`).

Для проверки корректности решения используется отрывок из примера с двумя сокращениями сердца:

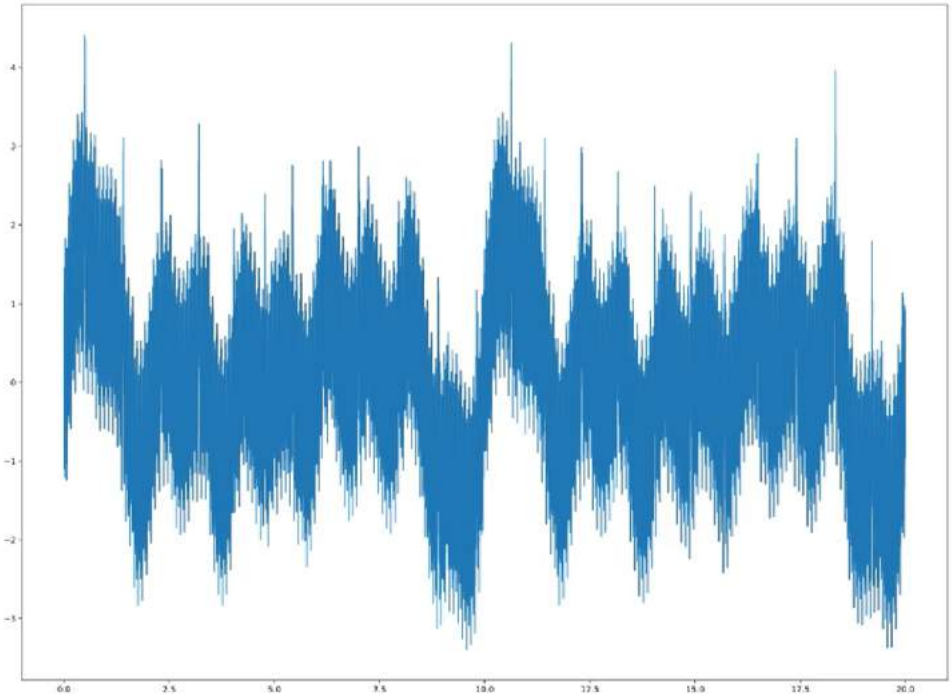


Пример №1

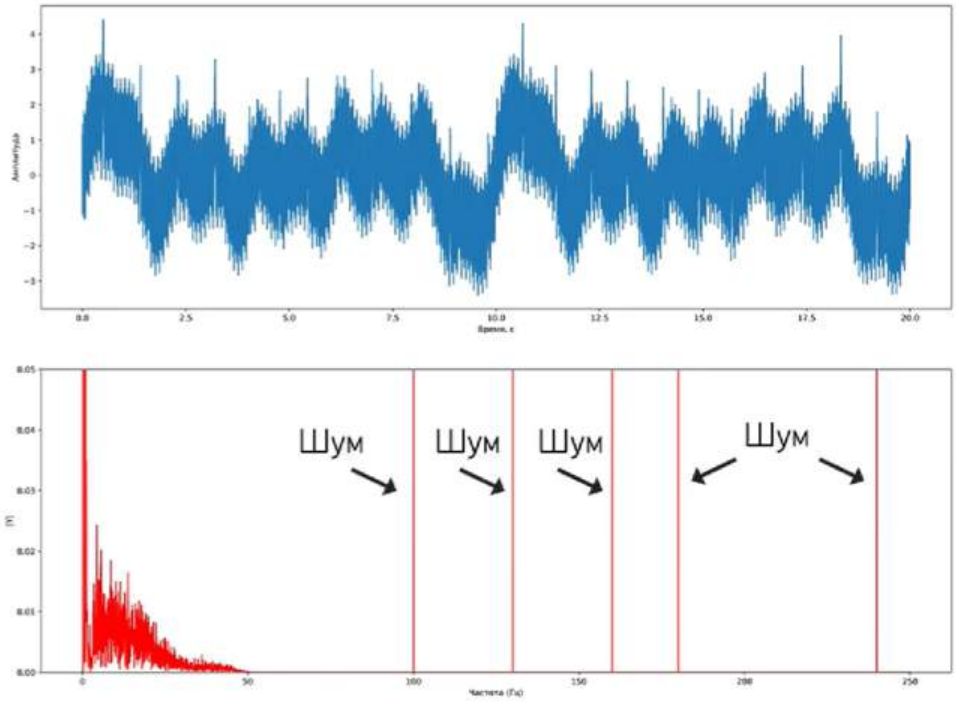
Стандартный ввод
-0.11821723252011541 1.135312534894712 -0.21982218822262528 -1.1057721743502749 -0.29103421006186825 1.4524076224589373 -0.30214821283080107 -0.4971958489822036 0.6206745987216689 -0.7334067500591215 0.5065983543323755 0.5871562045492225 ...
Стандартный вывод
2

Решение

Визуализируем пример из условия. Мы видим, что на записи присутствуют довольно сильные шумы, которые не позволяют сразу подсчитать число сердечных сокращений. От них нам нужно избавиться.



Проанализируем спектр данного сигнала. Для этого воспользуемся дискретным преобразованием Фурье из библиотеки numpy.




```
import matplotlib.pyplot as plt
import numpy as np

data = np.loadtxt('3.txt')

Fs = 500.0  # Частота оцифровки (дана в условии)
```

```
7       Ts = 1.0/Fs # Период оцифровки
8
9       n = len(data)
10      k = np.arange(n)
11      T = n/Fs
12      frq = k/T
13      frq = frq[range(n//2)]
14      Y = np.fft.fft(data)/n
15      Y = Y[range(n//2)]
16
17      plt.figure(figsize=(20,15))
18
19      # Строим график сигнала
20      plt.subplot(2,1,1)
21      plt.plot(np.arange(0,n)/Fs,data)
22      plt.xlabel('Время, с')
23      plt.ylabel('Амплитуда')
24
25      # Строим спектр сигнала
26      plt.subplot(2,1,2)
27      plt.plot(frq,abs(Y),'r')
28      plt.ylim(0,0.05)
29      plt.xlabel('Частота (Гц)')
30      plt.ylabel('|Y|')
31      plt.show()
```

На графике видны сильные шумы на частотах от 100 Гц и выше, а также на частотах порядка единиц Гц. Отфильтруем шумы с помощью полосового фильтра с полосой пропускания от 3 до 60 Гц.

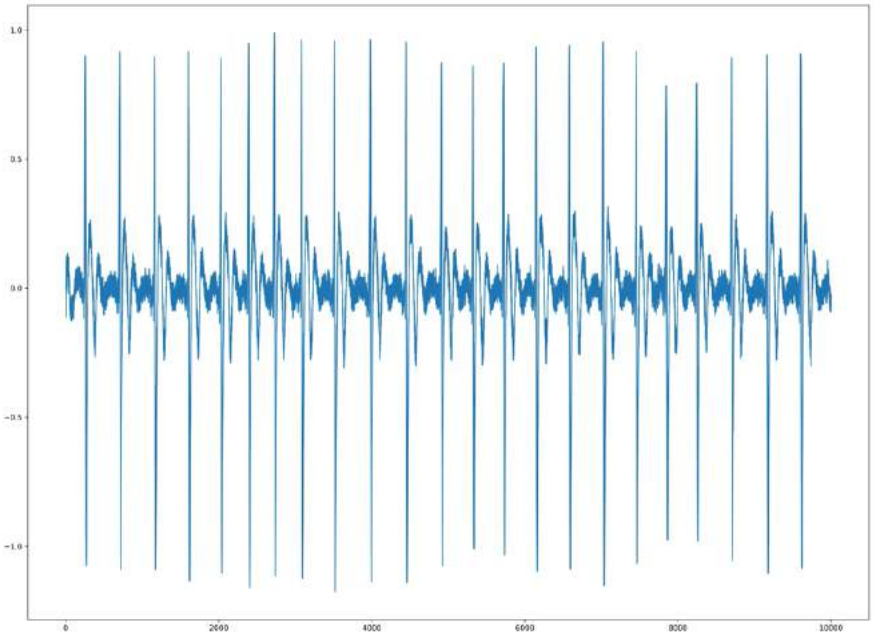
```
1       import matplotlib.pyplot as plt
2       import numpy as np
3       from scipy.signal import butter, lfilter
4
5       data = np.loadtxt('3.txt')
6
7       Fs = 500.0
8       nyq = 0.5 * Fs
9       low = 3 / nyq
10      high = 60 / nyq
11
12      b, a = butter(4, [low, high], btype='band')
13      filtered_data = lfilter(b, a, data)
14
15      plt.plot(filtered_data)
```

Импортируем методы butter и lfilter из scipy.signal [3 строка]. Запишим частоту Найквиста для нашего сигнала в переменую nyq [8 строка]. Верхнюю и нижнюю частоты полосы пропускания разделим на частоту Найквиста [9-10 строки] и запишем в переменные low и high. Деление границ полосы пропускания на частоту Найквиста обусловлено реализацией метода butter из библиотеки scipy. Коэффициенты фильтра Баттерворта вычисленные с помощью функции butter запишем в массивы b и a [12 строка]. Первый аргумент butter отвечает за порядок фильтра. В нашем случае возьмём его равным 4. Данное значение было подобрано экспериментально для фильтрации сигнала данного в примере. Увеличение порядка фильтра приводит к большей крутизне границ полосы пропускания, но может вызывать возникновение ложных выбросов сигнала. Значение band параметра dtype означает что нам

требуются коэффициенты именно полосового фильтра. [13 строка] Затем данные коэффициенты передаются в функцию filter, которая фильтрует сигнал, записанный в data. Результат фильтрации присваивается массиву filtered data.

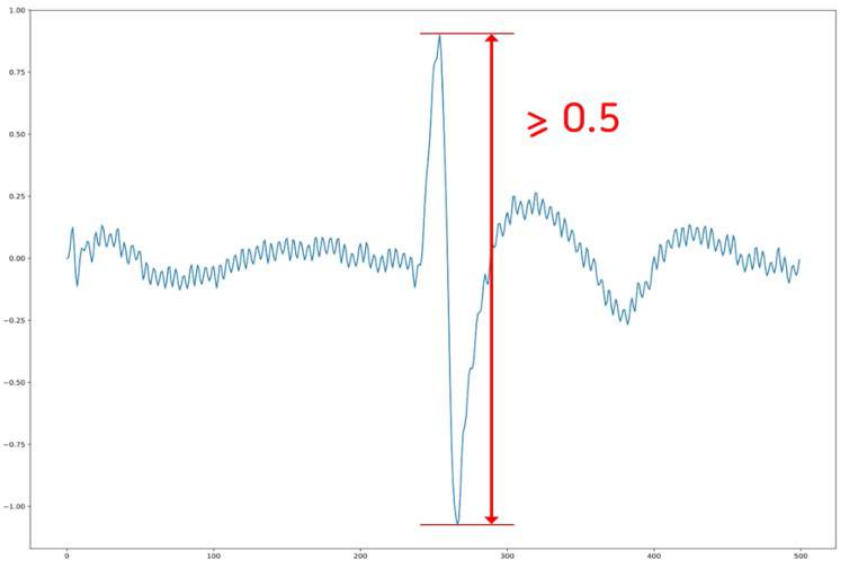
Таким образом, считав пример сигнала из данной задачи, мы визуализируем уже сигнал, пропущенный через полосовой фильтр. [15 строка].

Визуализировав отфильтрованный сигнал, мы видим, что он теперь пригоден для подсчёта числа сердечных сокращений.



Проверять наличие R-зубца мы будем, вычисляя амплитуду сигнала на отрезке.

Порог амплитуды подберём, рассмотрев отфильтрованный сигнал — в нашем случае возьмём его равным 0.5:



Далее осуществляем подсчет пиков:

```
from scipy.signal import butter, lfilter

data = list(map(float, input().split()))

nyq = 0.5 * 500
low = 3 / 250
high = 60 / 250
b, a = butter(4, [low, high], btype='band')
data = lfilter(b, a, data)

i = 0
beats = 0

while i < len(data) - 20:
    chunk = data[i:i+20]
    if max(chunk) - min(chunk) > 0.5:
        beats += 1
        i += 150
    i += 1

print(beats)
```

Считаем данные в переменную data.

Отфильтруем сигнал полосовым фильтром

Как и в предыдущих задачах, введём переменные i и beats, и как раньше будем подсчитывать число сердечных сокращений.

Единственное отличие будет заключаться в условии обнаружения R-зубца - в нашем случае будем считать, что отрезок содержит R-зубец если амплитуда сигнала на этом отрезке больше 0.5.

В конце выведем ответ.

3.2. Блок заданий 2

Задача 3.2.1. (10 баллов)

Для проведения различных исследований мозга иногда используют сигнал ЭЭГ. С помощью анализа данного сигнала можно выявить характерные состояния, например, закрытие глаз. При закрытии глаз у большинства людей на ЭЭГ усиливаются колебания в диапазоне от 8 до 13 Гц, называемые альфа-ритмом. Данные ЭЭГ, которые предстоит анализировать в этой задаче, взяты из архива PhysioBank. Они представляют собой последовательность целых чисел, лежащих в диапазоне от -440 до 300, разделённых запятыми и пробелами. Число перед запятой относится к первому каналу, число после - ко второму. Например, последовательность

1,2 3,4 5,6 7,8

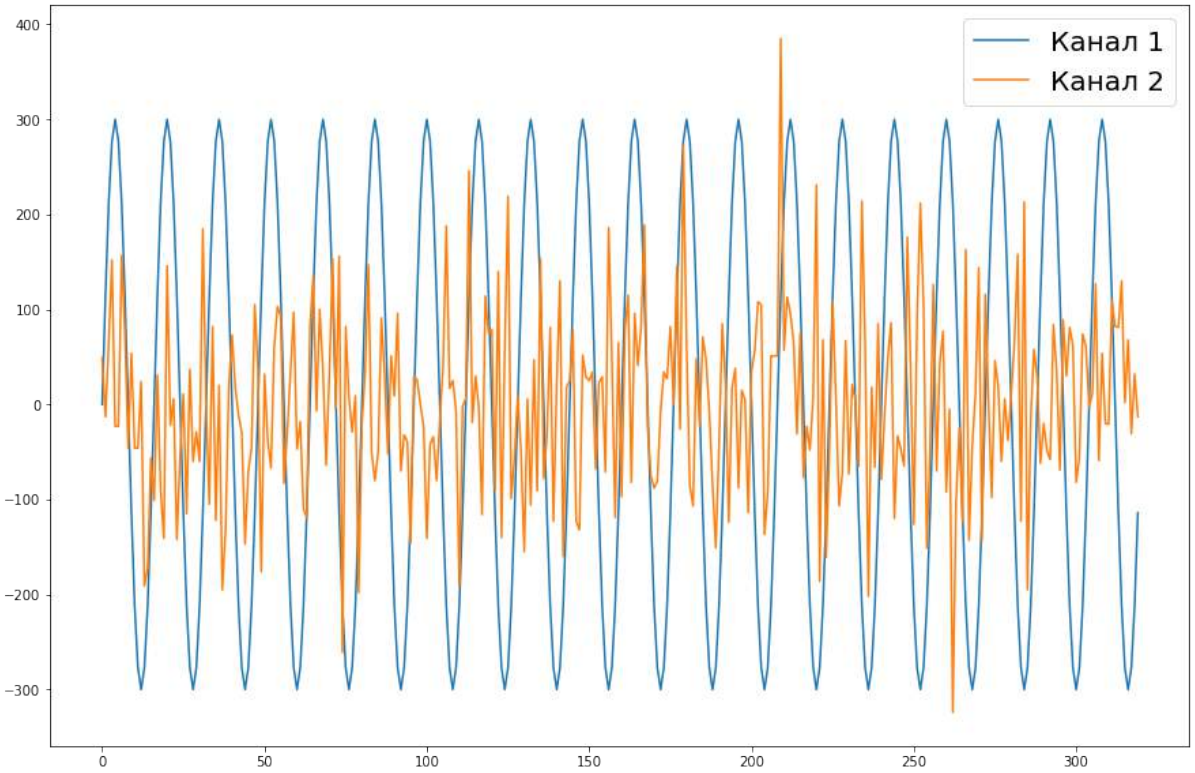
следует рассматривать как два сигнала: чила 1, 3, 5, 7 относятся к первому каналу, 2, 4, 6, 8 - ко второму. В тестовых данных требуется по одному каналу передаётся сигнал ЭЭГ, записанный с человека с открытыми глазами, по другому - с закрытыми. Сигнал был оцифрован с частотой 160 Гц.

Требуется определить номер канала, сигнал которого соответствует закрытым

глазам.

Пример входных данных, в которых ЭЭГ по второму каналу соответствует закрытым глазам доступен по ссылке (`https://www.dropbox.com/s/jow8pa0z348oa34/1.txt?dl=1`).

Для проверки корректности решения в качестве первого теста выступает последовательность чисел, у которых первому каналу соответствует синусоида с частотой 10 Гц, а второму - белый шум.



Пример №1

Стандартный ввод

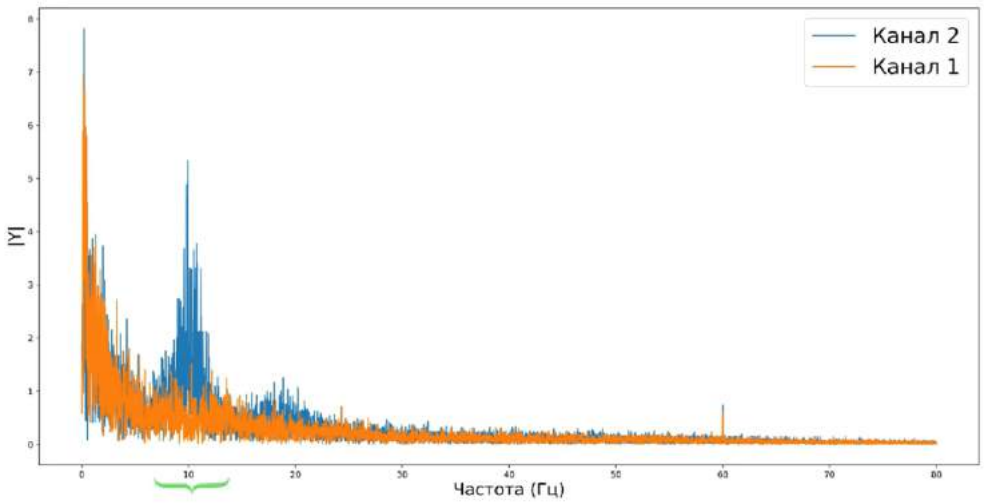
`0,49 114,-13 212,64 277,152 300,-23 277,-23 212,157`
`114,76 0,-46 -114,54 -212,-46 -277,-46 -300,24`
`...`

Стандартный вывод
`1`

Решение

В данной блока требуется определить какой из двух сигналов был записан при закрытых глазах. Для ответа на этот вопрос сравним спектры данных сигналов. При закрытии глаз у большинства людей на энцефалограмме увеличивается амплитуда гармоник в диапазоне от 8 до 13 Гц.

Спектры исходных сигналов:



Просуммируем амплитуду колебаний в этом диапазоне. Будем считать, что закрытым глазам соответствует сигнал, у которого данная сумма больше.

Приступим к написанию кода

```
import numpy as np

def get_spectrum(y):
    Fs = 160.0
    Ts = 1.0/Fs
    n = len(y)
    k = np.arange(n)
    T = n/Fs
    frq = k/T
    frq = frq[range(n//2)]
    Y = np.fft.fft(y)/n
    Y = Y[range(n//2)]
    return frq, abs(Y)

def get_alpha(y):
    frq, Y = get_spectrum(y)
    alpha = 0
    for freq, ampl in zip(frq, Y):
        if freq > 8 and freq < 13:
            alpha += ampl
    return alpha

st = [s.split(',') for s in input().split()]
sig1, sig2 = zip(*st)
sig1 = [int(s) for s in sig1]
sig2 = [int(s) for s in sig2]
alpha1 = get_alpha(sig1)
alpha2 = get_alpha(sig2)
print('1' if alpha1 > alpha2 else '2')
```

Импортируем библиотеку numpy для вычисления спектра сигнала

Напишем функцию get_spectrum, которая будет получать в качестве аргумента сигнал, для которого нужно вычислить его спектр.

По условию частота оцифровки сигнала составляет 160 Гц, запишем её в переменную Fs. Период оцифровки запишем в переменную Ts.

В переменную n запишем число значений в обрабатываемом сигнале.

В переменную k запишем массив значений от 0 до n-1. Он нам потребуется для вычисления частоты наших гармоник.

Зная частоту оцифровки, мы можем записать в массив frq частоты нашего сигнала

И затем выполнить дискретное преобразование Фурье. На 10 и 12 строках мы берём только половину значений массивов в силу симметричности получаемого после преобразования спектра.

Затем мы возвращаем два массива - массив частот frq и массив амплитуд Y, причём последний по модулю, так как после преобразования в нём хранятся комплексные числа.

Напишем функцию get_alpha, которая будет вычислять сумму амплитуд всех частот в спектре сигнала от 8 до 13 Гц. В качестве аргумента данная функция получает сигнал.

В массивы frq и Y сохраним результат работы функции get_spectrum.

Сумму амплитуд интересующих нас гармоник будем хранить в переменной alpha.

Будем последовательно перебирать элементы массивов frq и Y. На i-й позиции массива frq хранится частота i-й гармоники, а на i-й позиции массива Y хранится её амплитуда. Соответственно, если значение элемента первого массива лежит в диапазоне от 8 до 13, мы будем прибавлять соответствующее ему значение массива амплитуд в переменную alpha.

Таким образом функция будет возвращать амплитуду альфа-ритма.

Теперь остаётся реализовать считывание подаваемых на вход сигналов в разные списки и вычислить амплитуду альфа-ритма для каждого из этих сигналов. В выводим номер сигнала, у которого амплитуда альфа-ритма больше.

Задача 3.2.2. (12 баллов)

Сигналы ЭЭГ, которые предстоит проанализировать в данной задаче, взяты из архива PhysioBank. Значения сигнала разделены пробелом и представляют собой последовательность целых чисел, лежащих в диапазоне от -500 до 600. Частота оцифровки сигнала составляет 256 Гц.

К некоторым участкам оригинального сигнала были добавлены гармоники, лежащие в диапазоне частот альфа-ритма. Длительность изменённых участков составляет от 10 до 130 секунд, интервал между участками - от 30 до 210 секунд. Амплитуды и частоты добавленных гармоник одинаковы для всех тестовых сигналов.

Определите число участков с увеличенной амплитудой колебаний в диапазоне от 8 до 13 Гц. Пример тестового сигнала с 19 участками вы можете скачать по ссылке (`https://www.dropbox.com/s/9a7m34r2jff8mnf/2.txt?dl=1`). Данный сигнал подаётся на вход в качестве второго теста.

Примечание: излишние колебания некой величины (x) может помочь сгладить комплементарный фильтр и введение величины y, на t-ом шаге описываемой форму-

лой:

$$y[t] = (1 - k) \cdot y[t - 1] + k \cdot x[t]$$

Пример №1

Стандартный ввод
-47 -38 -36 -14 -3 -9 0 4 23 35 68 52 27 13 -8 -25 -32 -33 -33 -19 8 21 22 10 0 0 25 18 20 16 10 4 0 -4 -18 -6 15 31 34 54 57 44 ...
Стандартный вывод
1

Решение

Во второй задаче второго блока на записи электроэнцефалограммы требуется определить число участков с увеличенной амплитудой альфа-ритма. Для начала скачаем образец сигнала и начнём работать с ним.

Построим график амплитуды альфа-ритма в сигнале от времени. Для этого скопируем функции get_spectrum и get_alpha из предыдущей задачи. Единственное отличие будет заключаться в переменной Fs функции get_spectrum - в данной задаче частота оцифровки сигнала равна 256 Гц.

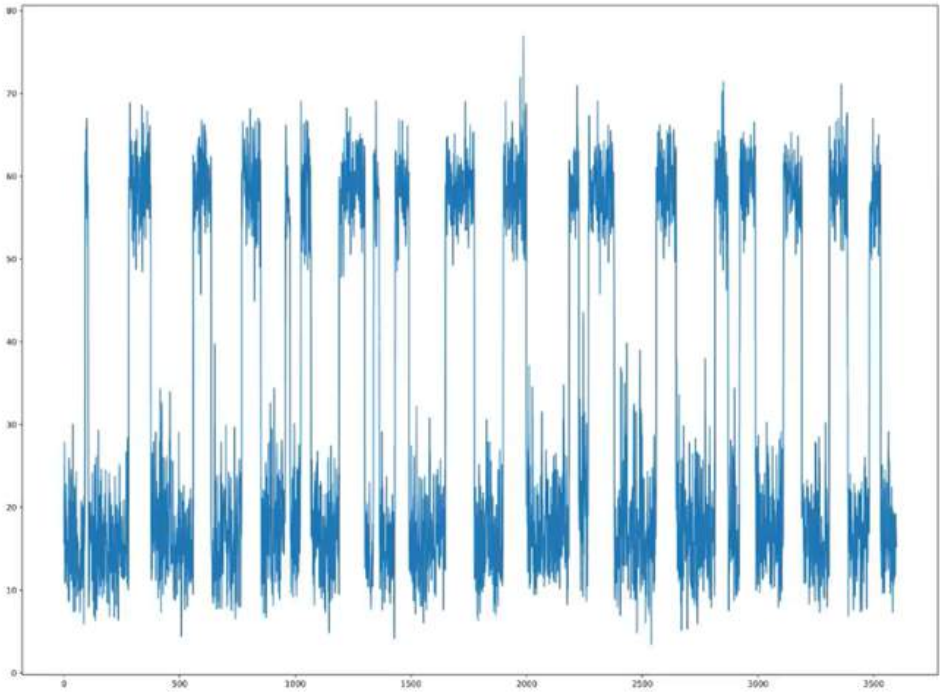
Сигнал запишем в массив data. Будем разбивать его на равные отрезкии и для каждого из них будем вычислять амплитуду альфа-ритма. Количество точек в отрезке сигнала возьмём равным 256 и запишем в переменную chunk_size.

В цикле будем перебирать значения переменной start которая будет являться началом отрезка. Амплитуды альфа-ритма будем вычислять с помощью функции get_alpha и хранить в списке alphas.

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3
4   def get_spectrum(y):
5       Fs = 256.0
6       Ts = 1.0/Fs
7       n = len(y)
8       k = np.arange(n)
9       T = n/Fs
10      frq = k/T
11      frq = frq[range(n//2)]
12      Y = np.fft.fft(y)/n
13      Y = Y[range(n//2)]
14      return frq, abs(Y)
15
16  def get_alpha(y):
17      frq, Y = get_spectrum(y)
18      alpha = 0
19      for freq, ampl in zip(frq, Y):
20          if freq > 8 and freq < 13:
21              alpha += ampl
22      return alpha
```

```
23
24      data = np.loadtxt('5.txt')
25      alphas = []
26      chunk_size = 256
27      for start in range(0,len(data)-chunk_size,chunk_size):
28          alphas.append(get_alpha(data[start:start+chunk_size]))
29
30      plt.figure(figsize=(20,15))
31      plt.plot(alphas)
```

Построим график полученных значений.



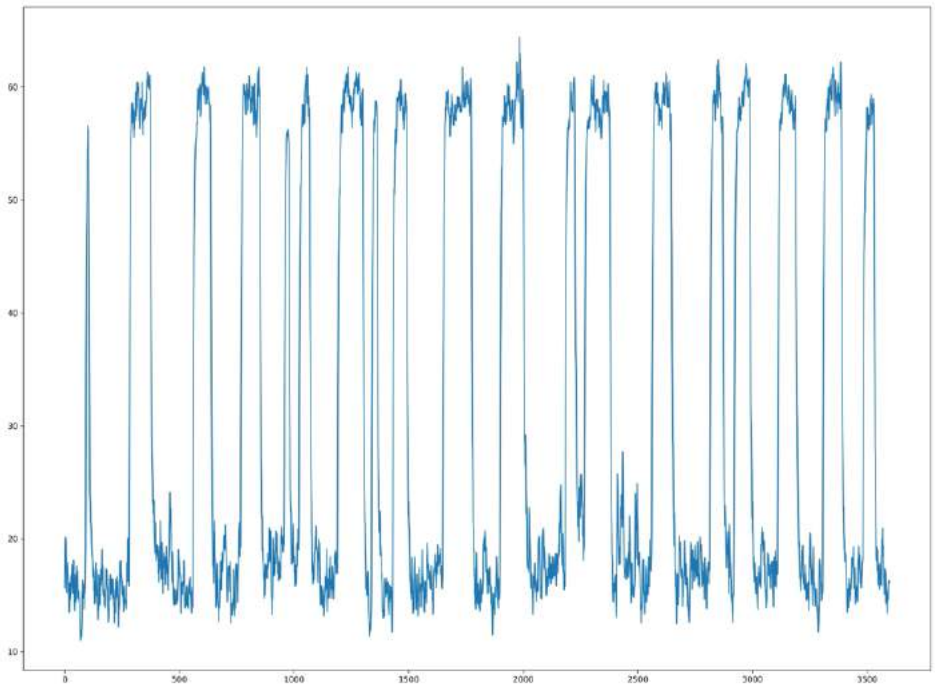
На графике мы видим 19 участков с увеличенной амплитудой альфа-ритма. Для упрощения их подсчёта попробуем сгладить график с помощью комплементарного фильтра

```
1       import numpy as np
2       import matplotlib.pyplot as plt
3
4       def get_spectrum(y):
5           Fs = 256.0
6           Ts = 1.0/Fs
7           n = len(y)
8           k = np.arange(n)
9           T = n/Fs
10          frq = k/T
11          frq = frq[range(n//2)]
12          Y = np.fft.fft(y)/n
13          Y = Y[range(n//2)]
14          return frq, abs(Y)
15
16      def get_alpha(y):
17          frq, Y = get_spectrum(y)
18          alpha = 0
19          for freq, ampl in zip(frq, Y):
```

```
20              if freq > 8 and freq < 13:
21                  alpha += ampl
22          return alpha
23
24      data = np.loadtxt('5.txt')
25      alphas = []
26      chunk_size = 256
27      for start in range(0,len(data)-chunk_size,chunk_size):
28          alphas.append(get_alpha(data[start:start+chunk_size]))
29
30      for i in range(1,len(alphas)):
31          alphas[i] = 0.8 * alphas[i-1] + 0.2 * alphas[i]
32
33      plt.figure(figsize=(20,15))
34      plt.plot(alphas)
```

Чем больше коэффициент перед i-1 элементом списка, тем сильнее будут сглажены резкие скачки уровня альфа-ритма.

После применения фильтра сигнал стал чище. Теперь подсчитаем число пиков альфа-ритма.



Введём переменную level [строка 32] равную среднему арифметическому максимального и минимального значений уровня альфа-ритма. В переменную peaks [строка 33] запишем число найденных пиков [строки 34-36]. Если из двух последовательных значений уровня альфа-ритма первое меньше значения переменной level а второе больше – будем считать что мы нашли пик [строка 37]. Наконец, выведем полученное число пиков.

```
1       import numpy as np
2
3       def get_spectrum(y):
4           Fs = 256.0
5           Ts = 1.0/Fs
6           n = len(y)
```

```
7           k = np.arange(n)
8           T = n/Fs
9           frq = k/T
10          frq = frq[range(n//2)]
11          Y = np.fft.fft(y)/n
12          Y = Y[range(n//2)]
13          return frq, abs(Y)
14
15      def get_alpha(y):
16          frq, Y = get_spectrum(y)
17          alpha = 0
18          for freq, ampl in zip(frq, Y):
19              if freq > 8 and freq < 13:
20                  alpha += ampl
21          return alpha
22
23      data = np.loadtxt('5.txt')
24      alphas = []
25      chunk_size = 256
26      for start in range(0,len(data)-chunk_size,chunk_size):
27          alphas.append(get_alpha(data[start:start+chunk_size]))
28
29      for i in range(1,len(alphas)):
30          alphas[i] = 0.8 * alphas[i-1] + 0.2 * alphas[i]
31
32      level = (max(alphas)+min(alphas))/2
33      peaks = 0
34      for i in range(1,len(alphas)):
35          if alphas[i-1]<=level and alphas[i]>level:
36              peaks += 1
37      print(peaks)
```

Для обработки сигнала, полученного через поток ввода, заменим 23 строку на

```
data = np.array([int(s) for s in input().split()])
```

Для проверки решений используется код на языке Python, представленный в Приложении 2.

Задача 3.2.3. (8 баллов)

Как и в предыдущей задаче, к сигналу ЭЭГ были добавлены гармоники, лежащие в диапазоне частот альфа-ритма. Кроме того, некоторые участки сигнала были заменены белым шумом значительной большей амплитуды.

Определите число участков с увеличенной амплитудой колебаний в диапазоне от 8 до 13 Гц, при этом не являющихся белым шумом.

Временные интервалы между изменёнными участками сигнала (в том числе и заменёнными шумом) и их продолжительность такие же, как и в предыдущей задаче. Частота оцифровки сигнала равна 256 Гц. Пример тестового сигнала с 5 промежутками с увеличенной амплитудой альфа-ритма, при этом не являющихся шумом, вы можете скачать по ссылке (`https://www.dropbox.com/s/ldvthnxm3m8685x/3.txt?dl=1`). Данный сигнал подаётся на вход в качестве второго теста.

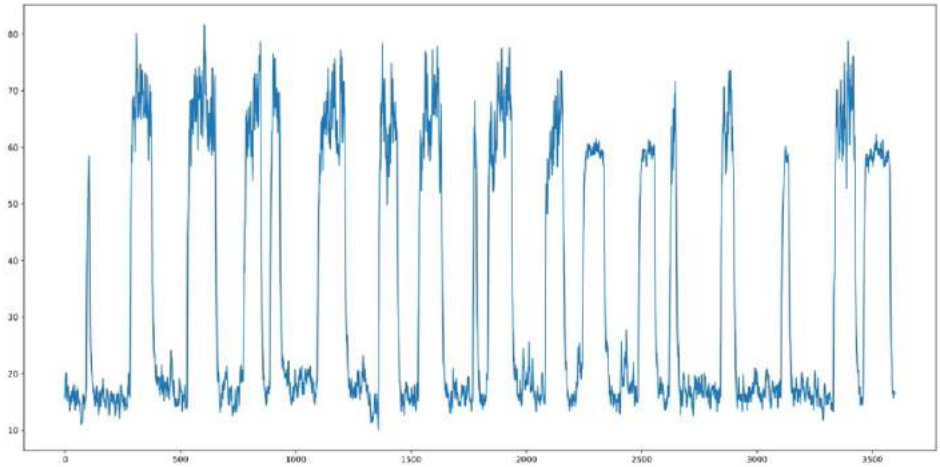
Пример №1

Стандартный ввод
-47 -38 -36 -14 -3 -9 0 4 23 35 68 52 27 13 -8 -25 -32 -33 -33 -19 8 21 22 10 0 0 25 18 20 16 10 4 0 -4 -18 -6 ...
Стандартный вывод
1

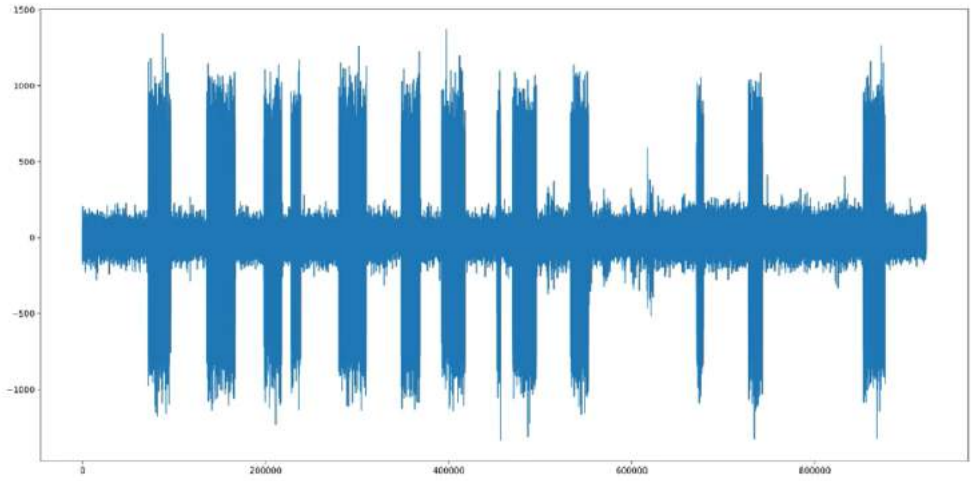
Решение

В третьей задаче второго блока на записи электроэнцефалограммы требуется определить число участков с увеличенной амплитудой альфа-ритма, при этом игнорируя участки с белым шумом.

Построим график уровня альфа-ритма для тестового сигнала. На нём мы видим 18 пиков, но в условии сказано, что сигнал содержит только 5 участков к которым были добавлены гармоники от 8 до 13 Гц, остальные пики альфа-ритма соответствуют участкам с белым шумом. Построим график сигнала



Построим график сигнала:



На нём мы отчётливо видим участки с белым шумом. Будем считать, что если

амплитуда сигнала больше 1300, то мы имеем дело с шумом. Решение этой задачи, за исключением пары строчек кода, полностью совпадает с решением предыдущей. В цикле мы дополнительно вводим переменную chunk. В ней будет хранится срез сигнала, по которому вычислялось значение alpha. Если амплитуда сигнала на этом отрезке превышает 1300, то мы не увеличиваем переменную beats, тем самым игнорируя участок с белым шумом.

```
import numpy as np

def get_spectrum(y):
    Fs = 256.0
    Ts = 1.0/Fs
    n = len(y)
    k = np.arange(n)
    T = n/Fs
    frq = k/T
    frq = frq[range(n//2)]
    Y = np.fft.fft(y)/n
    Y = Y[range(n//2)]
    return frq, abs(Y)

def get_alpha(y):
    frq, Y = get_spectrum(y)
    alpha = 0
    for freq, ampl in zip(frq, Y):
        if freq > 8 and freq < 13:
            alpha += ampl
    return alpha

data = np.array([int(s) for s in input().split()])
alphas = []
chunk_size = 256
for start in range(0,len(data)-chunk_size,chunk_size):
    alphas.append(get_alpha(data[start:start+chunk_size]))

for i in range(1,len(alphas)):
    alphas[i] = 0.8 * alphas[i-1] + 0.2 * alphas[i]

level = (max(alphas)+min(alphas))/2
peaks = 0
for i in range(1,len(alphas)):
    if alphas[i-1]<=level and alphas[i]>level:
        chunk = data[i*chunk_size:(i+1)*chunk_size]
        if max(chunk)-min(chunk) < 1300:
            peaks += 1
print(peaks)
```

3.3. Блок заданий 3

В данном блоке потребуется работать с библиотеками OpenCV, Scikit-Learn, Dlib. Так как решение подготавливается на собственных персональных компьютерах, данные библиотеки вы можете установить с помощью pip.

Полезная литература и ссылки:

Андреас Мюллер, Сара Гвидо, Введение в машинное обучение с помощью Python (2017)

www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

www.stackoverflow.com/questions/41912372/dlib-installation-on-windows-10

www.pythonworld.ru/osnovy/pip.html

www.cmake.org/install — может потребоваться установка cmake.

Задача 3.3.1. (6 баллов)

В первой задаче вам предстоит работать с данными по раку молочной железы Университета Висконсин. Они могут быть получены вызовом функции load_breast_cancer из модуля datasets библиотеки Scikit-Learn. В данных записаны клинические измерения опухолей молочной железы и указано является ли опухоль доброкачественной или злокачественной.

Ваша задача состоит в построении трёх моделей классификации опухолей с помощью библиотеки Scikit-Learn, которые на основании измерений опухоли будут прогнозировать её тип с точностью не менее 90%. К моделям предъявляются следующие требования:

- Модели должны храниться в списке models.
- Модели должны быть разных типов. Например, нельзя дважды использовать метод k ближайших соседей.
- Нельзя использовать классификатор опорных векторов (SVC).

Пример решения вы можете скачать по ссылке (https://goo-gl.ru/4TAk). Данное решение получило бы 0 баллов, так как все три модели этого решения - это классификатор опорных векторов, но оно демонстрирует, как именно должен выглядеть ответ.

Решение будет тестироваться с помощью этого (https://goo-gl.ru/4TAs) кода (random_state будет другим).

Решение скопируйте в текстовое поле, после чего оно будет протестировано разработчиками Олимпиады НТИ. Оценки будут выставлены после проверки. За каждую модель, определяющую тип опухоли с точностью не ниже 90% и удовлетворяющую условиям, вы можете получите по 2 балла. Всего за эту задачу можно получить 6 баллов.

Обратите внимание, что у вас есть только одна попытка для ввода ответа.

Решение

В первой задаче третьего блока требуется построить три модели для классификации опухолей с помощью библиотеки Scikit-Learn, которые на основании измерений опухоли будут прогнозировать её тип с точностью не менее 90

Целью данного задания является познакомить участников с библиотекой машинного обучения Scikit-Learn. Рассмотрим для примера три следующих модели:

- логистическая регрессия,
- метод k-ближайших соседей,
- случайный лес.

Все три модели с параметрами по умолчанию [строки 7-9] дают достаточную точность на доступном для участников тестовом коде. Уже в таком виде задание выполнено на максимальный балл:

```
1   from sklearn.linear_model import LogisticRegression
2   from sklearn.neighbors import KNeighborsClassifier
3   from sklearn.ensemble import RandomForestClassifier
4
5   models=[None,None,None]
6
7   models[0] = LogisticRegression(random_state=0)      # 95.1%
8   models[1] = KNeighborsClassifier()                  # 93.0%
9   models[2] = RandomForestClassifier(random_state=0)  # 95.1%
```

Улучшим точность каждого из классификаторов.

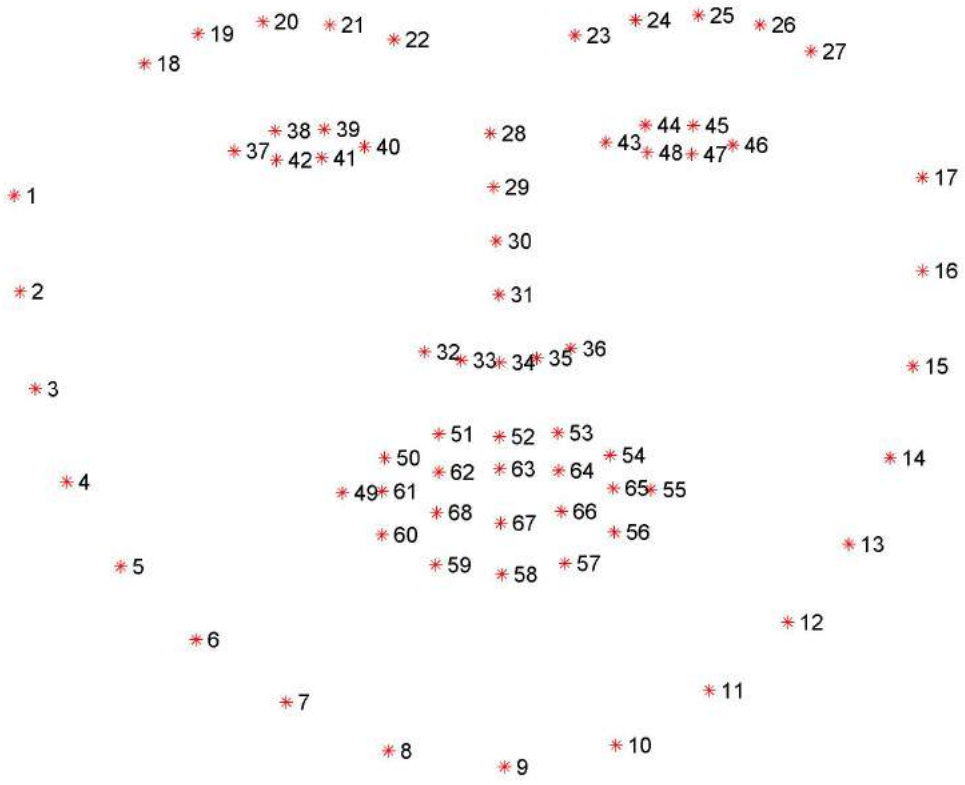
Добавим к логистической регресии и методу k-ближайших соседей MinMaxScaler. Изменим солвер у логистической регрессии. Точность двух классификаторов возрастёт до 97.2%. Изменение параметра criterion у случайного леса увеличит точность до 95.6%.

```
1   from sklearn.linear_model import LogisticRegression
2   from sklearn.neighbors import KNeighborsClassifier
3   from sklearn.ensemble import RandomForestClassifier
4   from sklearn.preprocessing import MinMaxScaler
5   from sklearn.pipeline import Pipeline
6
7   models=[None,None,None]
8
9   models[0] = Pipeline([('scaler', MinMaxScaler()),   # 97.2%
10            ('logreg', LogisticRegression(solver='newton-cg',
11                                          random_state=0))])
12  models[1] = Pipeline([('scaler', MinMaxScaler()),   # 97.2%
13                        ('estimator',KNeighborsClassifier())])
14  models[2] = RandomForestClassifier(criterion='entropy',
15                                     random_state=0)  # 95.6%
```

Вот так, за счёт небольших изменений, мы увеличили точность моделей.

Задача 3.3.2. (8 баллов)

Для выделения ключевых точек лица с фотографии применяют библиотеку Dlib. Для того, чтобы выделить 68 ключевых точек, следует воспользоваться предобученной моделью хранящейся по ссылке (`https://www.dropbox.com/s/9x6codhj1omkmd2/shape_predictor_68_face_landmarks.dat?dl=1`). Ключевые точки лица нумеруются следующим образом (см. рис. ниже):



Ширину открытия глаз будем считать как усреднённое расстояние между следующими парами точек: между №38 и №42, между №39 и №41, №44 и №48, №45 и №47, поделённое на расстояние межу точками №9 и №28. В данной задаче от вас требуется найти в наборе из 50 фотографий те фотографии, на которых ширина открытия глаз максимальная и минимальная.

Замечание. При определении координат ключевых точек лица может возникнуть неоднозначность, поэтому:

1. При поиске лица используйте детектор лиц со вторым аргументом равным 1, как в примере (`http://dlib.net/face_detector.py.html`).
2. Ищите лицо и ключевые точки на изображении в градациях серого, т.е. если image - считанное изображение, то применяйте детекторы лиц и ключевых точек к изображению gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).

Вам предстоит работать с изображениями из набора данных, собранного Маркусом Вебером в Калифорнийском технологическом институте. Архив с изображениями необходимо скачать по ссылке (`https://www.dropbox.com/s/h3tjg55hdm65cth/faces.zip?dl=1`). Изображения в архиве носят названия N.jpg - где N - натуральное число от 1 до 431 включительно. После нажатия кнопки "Нажмите чтобы начать решать"для вас будет сгенерирован файл, содержащий 50 натуральных чисел, разделённых пробелом. Числа в данном файле - номера фотографий, среди которых вы должны будете за 5 минут найти две фотографии с максимальной шириной открытия глаз и минимальной шириной открытия глаз. Ответ напишите в появившемся текстовом поле в виде "N1 N2"(без кавычек), где N1 - название фото (без .jpg), на котором ширина открытия глаз максимальная, а N2 - минимальная. Образец ввода и вывода дан ниже.

Внимание! У вас будет всего одна попытка. Прежде чем нажимать "Нажмите чтобы начать решать"убедитесь, что вы уверенно можете в течение 5 минут найти

из произвольного набора 50 фотографий две требуемые.

Пример №1

Стандартный ввод
276 345 7 75 93 157 379 247 188 387 89 18 376 149 283 218 254 368 338 291 299 272 381 163 405 50 70 343 104 60 142 401 35 132 192 417 85 367 403 130 309 239 144 125 305 212 385 183 194 235
Стандартный вывод
183 163

Решение

Во данной задаче требуется найти из случайного набора фотографии те, на которых ширина открытия глаз максимальная и минимальная.

Сначала вычислим ширину открытия глаз для всех фотографий и запишем эти данные в один текстовый файл. Для этого напишем первую программу.

Импортируем все необходимые библиотеки и загрузим предварительно обученные модели для детекции лиц и ключевых точек. Ширину открытия глаз запишем в файл distances. Названия фотографий будем перебирать в цикле, а загружать изображения будем с помощью функции imread библиотеки OpenCV.

Сделаем изображение чёрно-белым и найдём на нём лица с помощью детектора лиц.

Так как у нас на всех изображениях присутствует только одно лицо, то в массиве rects только один элемент.

Поэтому при поиске ключевых точек мы обращаемся элементу rects с индексом 0. После применения метода из модуля fase_utils в shape хранятся координаты ключевых точек лица.

Далее мы можем рассчитать ширину открытия глаз. На присутствующем в условии задачи изображении нумерация точек начинается с единицы, т.е. 38й точке о которой шла речь в условии соответствует элемент массива shape с индексом 37. Расстояние между точками мы вычисляем с помощью функции norm модуля linalg библиотеки numpy

Затем мы записываем в файл distances строку содержащую номер изображения и ширину открытия глаз.

```
1     import dlib
2     import cv2
3     from numpy.linalg import norm
4     from imutils import face_utils
5     from dlib import shape_predictor
6
7     detector = dlib.get_frontal_face_detector()
8     predictor = shape_predictor('shape_predictor_68_face_landmarks.dat')
9     with open('distances.txt', 'w') as file:
10        for i in range(1,432):
11            name = 'faces/{}.jpg'.format(i)
12            image = cv2.imread(name)
```

```
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            rects = detector(gray, 1)
            shape = predictor(gray, rects[0])
            shape = face_utils.shape_to_np(shape)
            dist_eyes = norm(shape[37] - shape[41]) +\
                        norm(shape[38] - shape[40]) +\
                        norm(shape[43] - shape[47]) +\
                        norm(shape[44] - shape[46])
            dist_eyes /= norm(shape[8] - shape[27])

            file.write('{}\t{}\n'.format(i,dist_eyes))
```

Теперь у нас есть файл, содержащий всю необходимую для решения задачи информацию. Напишем программу, выдающую номера фотографий с максимальной и минимальной шириной открытия глаз.

Считаем содержимое файла distances в словарь dists. Ключом словаря является номер изображения, значением - ширина открытия глаз.

Строку из 50 чисел, определяющую набор изображений из которых нужно найти фотографии с минимальной и максимальной шириной открытия глаз будем хранить в текстовом файле samples. Считаем строку в список с тем же названием.

В переменной max_name будем хранить название изображения с максимальной шириной открытия глаз, в переменной max_val — значение, соответствующее данной фотографии. Аналогично для минимальной ширины введём переменные min_name и min_val. По умолчанию мы инициализируем их первым элементом списка samples.

Перебрав значения списка samples найдём требуемые изображения.

В конце выведем ответ.

```
with open('distances.txt','r') as file:
    dists = [(l.split()[0],
            float(l.split()[1])) for l in file.readlines()]
    dists = dict(dists)

with open('samples.txt','r') as file:
    samples = file.readline().split()

max_name, max_val  = samples[0], dists[samples[0]]
min_name, min_val  = samples[0], dists[samples[0]]

for sample in samples:
    if max_val < dists[sample]:
        max_val = dists[sample]
        max_name = sample
    if min_val > dists[sample]:
        min_val = dists[sample]
        min_name = sample

print(max_name, min_name)
```

Если записать в файл samples строку из примера, наша программа выведет числа 183 и 163, что совпадает с ответом из примера. Убедившись, что программа работает верно, мы можем теперь использовать её на любом другом случайном наборе фотографий.

Задача 3.3.3. (8 баллов)

В архиве по данной ссылке находятся фотографии 200 разных людей. Для каждого человека есть две фотографии: на одной он улыбается, а на другой - нет. Названия фотографий с улыбкой оканчиваются 'b', без улыбки — 'a'. Вам требуется написать класс Solver обладающий следующими свойствами:

- Конструктор класса должен принимать один аргумент - путь к папке, содержащей фотографии для обучения модели;
- Класс должен иметь метод predict, получающий в качестве аргумента путь к фотографии человека, и возвращающий False, если человек на фотографии не улыбается, и True, если улыбается.

При проверке файл с вашим кодом импортируется и создаётся экземпляр класса Solver. В качестве аргумента конструктор класса получает путь к директории, содержащей 200 случайных фотографий из архива. Названия фотографий будут изменены (только нумерация, буквы 'a' и 'b' останутся на месте), поэтому ознакомьтесь с работой os.listdir. Среди этих фотографии одинаковое количество изображений с улыбкой и без. Затем в цикле вызывается метод predict, которому в качестве аргумента последовательно подаётся путь к каждой из оставшихся фотографий. Названия этих фотографий не будут содержать букв 'a' и 'b'. Количество баллов за данное задание вычисляется по формуле:

$$16 \cdot \frac{score - 50}{50}$$

где score — процент изображений, для которых верно было определено наличие или отсутствие улыбки. Пример решения представлен ниже (`https://www.dropbox.com/s/dcgqwcp7lzws6sj/solution.py?dl=1`).

```
import os
import cv2
import dlib

class Solver:
    def __init__(self, path_to_dir):
        self.predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
        for file in os.listdir(path_to_dir):
            file_name = os.path.join(path_to_dir,file)
            image = cv2.imread(file_name)

    def predict(self, path_to_file):
        return False
```

В конструкторе класса создаётся поле predictor, в которое загружается предобученная модель. Затем в цикле последовательно перебираются все фотографии из директории, путь к которой был подан конструктору в качестве аргумента. Фотографии по очереди загружаются в переменную image, больше с ними ничего не происходит. В метод predict всегда возвращает False.

К вашему коду предъявляются следующие требования:

- Путь к изображениям должен формироваться с использованием os.path.join. Запрещена конкатенация строк вида

```
file_name = os.path.join + '/' + file
```

- Файл с предобученной моделью для определения ключевых точек хранится в той же директории, в которой будет находится ваш код, т.е. к нему можно обращаться напрямую shape_predictor_68_face_landmarks.dat. Файл тот же, что и в предыдущей задаче.
- Все изображения, которые будут использованы при оценке точности решения, присутствуют в архиве. Может возникнуть соблазн создать структуру вида ключ - значение, где в качестве ключа используется один или несколько параметров изображения, однозначно его идентифицирующих, а в качестве значения - ответ, к какому классу данное изображение относится, и использовать данную структуру в методе predict. Все подобные решения будут оценены на 0 баллов.
- Можно использовать любые стандартные библиотеки Python, библиотеки OpenCV, Dlib, Imutils, Scikit-Learn, Numpy, Scipy.

Проверить, что ваше решение будет протестировано корректно можно с помощью следующего кода (`https://goo-gl.ru/4TAL`):

```
import solution
import os

dir_with_images = 'smiles'
test_image = '1a.jpg'
sol = solution.Solver(dir_with_images)
print(sol.predict(os.path.join(dir_with_images,test_image)))
```

Предполагается, что файл с вашим кодом называется solution.py и находится с test.py в одной директории. Также в этой директории находится папка с изображениями smile и она содержит изображение 1a.jpg. Если test.py выполняется без ошибок и выводит True или False (в зависимости от того, как хорошо работает ваша программа), то решение будет протестировано корректно. Ограничение по времени на работу конструктора 5 минут, на работу метода predict - 10 секунд. Решение будет запускаться на CPU Core i5-M450 с 4 Гб оперативной памяти под управлением Ubuntu 16.04.5.

Решение скопируйте в текстовое поле, после чего оно будет протестировано разработчиками Олимпиады. Баллы будут выставлены после проверки.

Решение

В данной задаче требуется написать код определяющий улыбается ли человек на изображении.

В решении мы будем использовать координаты ключевых точек лица, получаемые с помощью библиотеки Dlib. Их мы будем подавать в качестве признаков для классификатора опорных векторов или SVC реализованного в библиотеке Scikit-Learn.

Импортируем все необходимые модули.

Напишем класс solver [строка 11], у которого конструктор получает в качестве аргумента путь к директории, содержащей изображения для обучения [строка 12].

И есть метод predict, который выводит выводит False если человек на изображении, путь к которому был передан в качестве аргумента, не улыбается и True если улыбается.

Как и предыдущей задаче, загрузим предобученную модель [строки 13-14].

А затем создадим конвейер, который будет масштабировать данные с помощью MinMaxScaler и классифицировать их используя метод опорных векторов. О том, как мы подобрали параметры классификатора, поговорим позднее.

В список data будем записывать координаты ключевых точек, а в список target - улыбается ли человек на изображении.

В переменной file будут перебираться названия всех файлов, хранящихся в папке, поданной конструктору в качестве аргумента [строка 19].

В список data добавляем результат работы функции get_shape. Именно в этой функции мы будем считывать изображение и находить координаты ключевых точек.

Изображения, на которых люди не улыбаются содержат в названии букву ‘a’, а те, на которых люди улыбаются букву ‘b’. Последние 4 символа названия изображения - это точка и разрешение jpg, буква ‘a’ или ‘b’ - 5 символ с конца. Добавляем в target 0, если человек на изображении не улыбается и 1 если улыбается [строка 22].

Для подачи обучающих данных в классификатор переводим их в массив numpy. [строки 23-24].

В конце работы конструктора обучаем модель [строка 25].

Теперь рассмотрим функцию get_shape [строка 27].

Считаем изображение путь, к которому был передан функции в качестве аргумента и найдём координаты ключевых точек лица.

В конце работы функции вернём кординаты точек в виде одномерного массива. Для этого воспользуемся методом flatten.

В функции predict сначала получим координаты ключевых точек для изображения, путь к которому был передан функции в качестве аргумента.

Воспользуемся предобученной моделью для определения типа изображения. Метод predict работает с массивом векторов признаков, поэтому используем метод reshape.

Predict возвращает массив. Мы подавали ему только один пример, поэтому predict_list содержит всего один элемент, к которому мы и обращаемся. Если этот элемент 0, то он будет приведён к значению булевого типа False, а если 1 - то к True, как того и требует условие задачи.

Данный код выдаст на тестовых данных точность порядка 95%.

```
1     import os
2     import cv2
3     import dlib
4     import numpy as np
5     from imutils import face_utils
6     from sklearn.svm import SVC
7     from sklearn.preprocessing import MinMaxScaler
8     from sklearn.pipeline import Pipeline
9     from dlib import shape_predictor as sh_p
10
11    class Solver:
```

```
12          def __init__(self, path_to_dir):
13              self.predictor = sh_p('shape_predictor_68_face_landmarks.dat')
14              self.detector = dlib.get_frontal_face_detector()
15              self.clf = Pipeline([('scaler', MinMaxScaler()),
16                           ('svm', SVC(C=10, kernel='linear'))])
17              data = []
18              target = []
19              for file in os.listdir(path_to_dir):
20                  file_name = os.path.join(path_to_dir,file)
21                  data.append(self._get_shape(file_name))
22                  target.append(0 if file[-5] == 'a' else 1)
23              data = np.array(data)
24              target = np.array(target)
25              self.clf.fit(data,target)
26
27          def _get_shape(self, file_name):
28              image = cv2.imread(file_name)
29              gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
30              rects = self.detector(gray, 1)
31              shape = self.predictor(gray, rects[0])
32              shape = face_utils.shape_to_np(shape)
33              return shape.flatten()
34
35          def predict(self, file_name):
36              shape = self._get_shape(file_name)
37              predict_list = self.clf.predict(shape.reshape(1,-1))
38              return bool(predict_list[0])
```

Теперь рассмотрим, как мы подобрали параметры модели.

Импортируем необходимые модули. Обратите внимание на 4 строку. Здесь мы импортируем класс GridSearchCV который осуществляет решётчатый поиск оптимальных параметров.

Данные загрузим из текстового файла ML.txt [строка 7]. В это файл мы предварительно записали координаты ключевых точек и классы, к которым относятся фотографии, т.е. человек на изображении с улыбкой или без.

[строка 8]. В X мы запишем вектора признаков, а в y — классы к которым относятся наши примеры [строки 9-10].

Точность нашей модели мы будем проверять на значениях параметров из списка param_grid.

С помощью GridSearchCV мы можем сравнить точность нашей модели для разных комбинаций параметров и вывести параметры, с которыми модель показала наибольшую точность.

```
1       from sklearn.svm import SVC
2       from sklearn.preprocessing import MinMaxScaler
3       from sklearn.pipeline import Pipeline
4       from sklearn.model_selection import GridSearchCV
5       import numpy as np
6
7       data = np.loadtxt('ML.txt')
8       X,y = data[:,:-1],data[:,-1]
9       model = Pipeline([('scaler', MinMaxScaler()),
10                        ('svm', SVC())])
11      param_grid = [
12      {'svm__C': [1, 10, 100, 1000],
```

```
    'svm__kernel': ['linear']},
    {'svm__C': [1, 10, 100, 1000],
    'svm__gamma': [0.001, 0.0001],
    'svm__kernel': ['rbf']},
    ]
    clf = GridSearchCV(model, param_grid, cv=10)
    clf.fit(X, y)
    print(clf.best_params_)
```

3.4. Блок заданий 4

Задача 3.4.1. (10 баллов)

Вам необходимо прорешать задачи курса "Цифровая электроника с Arduino", доступного по ссылке: `https://stepik.org/course/50222/syllabus`.

Система оценки

Баллы будут начислены после закрытия модуля пропорционально баллам, набранным в указанном курсе, то есть если участник команды решит 70%, будет выставлено 7 баллов.

3.5. Приложение. Справочные материалы

Для успешного выполнения задач данного блока вам необходимо познакомиться с библиотекой matplotlib:

`https://pythonworld.ru/novosti-mira-python/scientific-graphics-in-python.html`

Для наших целей будет достаточно использования интерфейса pyplot, рекомендуем ознакомиться с примером построения графиков на сайте: `https://matplotlib.org/`

В последней задаче будет использовано преобразование Фурье:

`https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%...`

`https://habr.com/post/196374/`

`https://habr.com/post/269991/`

Справка по функции преобразования Фурье в Python:

`https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.fft.fft.html#numpy.fft.fft`

`https://habr.com/post/269991/`

`https://plot.ly/matplotlib/fft/`

Фильтрация сигналов средствами scipy:

`https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html`

`https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html`

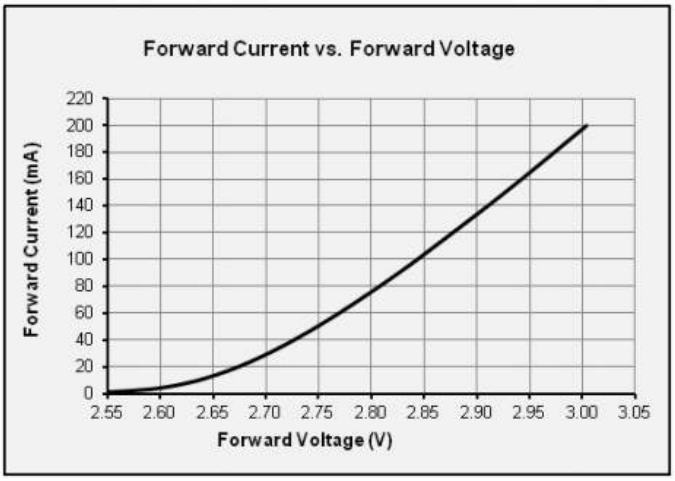
Ввод исходных данных в наших задачах осуществляется через stdin, проверка решения происходит с помощью stdout. Подключение всех необходимых стандартных библиотек нужно осуществлять самостоятельно.

Цифровая электроника с Arduino

4.1. Основы электроники

Задача 4.1.1. (3 балла)

В качестве фар машинки-робота использован осветительный светодиод LM561C, запитываемый от 2-х баночного LiPo аккумулятора (7.4 В). Последовательно со светодиодом, в цепь также включен ограничительный резистор. Используя приведенную ниже вольт-амперную характеристику светодиода, выберите номинал резистора так, чтобы ток через светодиод был по возможности ближе к 100 мА, но не превышал этого значения.



а. Выберите номинал ограничительного резистора из стандартного ряда номиналов:

1. 33 Ом
2. 39 Ом
3. 47 Ом
4. 56 Ом
5. 68 Ом
6. 82 Ом
7. 100 Ом
8. 120 Ом

б. Максимальная рассеиваемая мощность этого резистора не должна быть менее чем

1. 0.125 Вт
2. 0.25 Вт
3. 0.5 Вт
4. 0.75 Вт
5. 1 Вт
6. 2 Вт

Решение

В этой задаче вам надо понимать закон Ома и знать, что такое вольт-амперная характеристика.

В отличие от, например, резисторов, светодиоды являются нелинейными электронными компонентами. Ток через светодиод почти не протекает при малых значениях напряжения, находится в рабочем диапазоне в очень узком диапазоне напряжений, а при дальнейшем увеличении напряжения резко нарастает, до перегрева и выхода светодиода из строя. График зависимости тока от напряжения называется вольт-амперной характеристикой устройства.

Как видно из графика, требуемый ток в 100 мА получается при напряжении на диоде 2.85 В. Однако же на входе у нас 7.4 В, поэтому из них $U_R = 7.4 - 2.85 = 4.55$ В должны падать на ограничительном резисторе. Поскольку резистор и светодиод соединены последовательно, ток через резистор составляет те же 100 мА. По закону Ома:

$$R = U_R / I = 4.55 \text{ В} / 0.1 \text{ А} = 45.5 \text{ Ом}$$

Однако резисторы изготавливаются определенных номиналов, и Stepik предлагает выбрать один из них (33, 39, 47, 56, 68 Ом). Выбираем резистор с ближайшим большим значением - 47 Ом.

Какова же должна быть максимальная рассеиваемая мощность этого резистора? Мощность вычисляется по формуле:

$$P = U \cdot I = 4.55 \text{ В} \cdot 0.1 \text{ А} \approx 0.45 \text{ Вт}$$

Подбираем ближайшее большее значение (0.125, 0.25, 0.5, 0.75, 1, 2 Вт): 0.5 Вт.

Ответ: а - 3, б - 3.

4.2. Программируем Arduino: простые схемы и задачи

Задача 4.2.1. Простой blink (2 балла)

В этом шаге вам нужно будет изменить и проверить работу простейшего примера blink из стандартного набора примеров Arduino IDE, для этого следует изменить номер пина и интервал мигания, как указано в шаблоне!

Такие задачи стали возможны благодаря мини-симулятору Arduino, который имитирует действие некоторых функций библиотеки Arduino, изменяя состояние хранимых в памяти "пинов"и сообщая о каждом изменении платформе Stepik, чтобы

проверить правильность вашего решения. Вы пишете код КАК БЫ для Ардуино, на самом деле он выполняется в Степике, взаимодействуя с платформой через стандартный ввод-вывод, как и любая другая задача на программирования.

Вам НЕ НАДО трогать что-нибудь в шаблоне, кроме функций setup() и loop(). Разумеется, для решения более сложных задач могут понадобиться дополнительные функции или переменные, которые вы можете определить там же. Можно использовать большинство стандартных библиотек C++, но никакие библиотеки, специфические для Arduino, применить не получится. Мини-симулятор настраивается под задачу, и в нем, как правило, не будут работать никакие функции, не относящиеся конкретно к данной задачи.

Функции библиотеки Arduino, которые можно использовать в этой задаче:

void digitalWrite(int pin, int state): изменяет хранимое в памяти состояние пинов, и сообщает в Stepik о каждом изменении, вместе с отметкой времени. Если был задан неверный номер пина, либо этот пин не был переведен в режим OUTPUT, либо состояние пина не изменилось, то ничего не происходит.

void pinMode(int pin, int mode): устанавливает режим работы пина: INPUT, OUTPUT или INPUT_PULLUP.

void delay(int n): "Задержка"на n миллисекунд, а на самом деле - просто увеличение внутреннего счетчика времени. Мини-симулятор следит за временем, хотя, в отличие от реального микроконтроллера, счетчик времени увеличивается только и исключительно вызовами функции delay(). Любые операции, между которыми нет delay(), считаются выполняющимися мгновенно.

Если вы хотите испытать свой код на реальной схеме с Ардуино, просто перенесите в среду Ардуино написанный вами код и он, МОЖЕТ БЫТЬ, даже будет работать. Разумеется, вам сначала придется собрать соответствующую описанию схему.

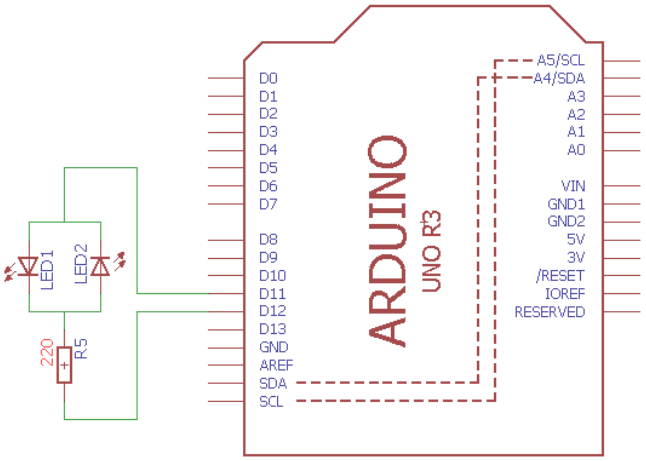
Пример программы-решения

Ниже представлено решение на языке C++

```
int LED_PIN;   // каким пином мигаем
int ON_PERIOD; // время свечения, мс
int OFF_PERIOD; // интервал между миганиями, мс
void setup()
{
    pinMode(LED_PIN,OUTPUT);
}
void loop()
{
    digitalWrite(LED_PIN, HIGH);
    delay(ON_PERIOD);
    digitalWrite(LED_PIN, LOW);
    delay(OFF_PERIOD);
}
```

Задача 4.2.2. Полицейский маячок (BLINK со встречным подключением светодиодов) (3 балла)

Два светодиода, красный и зеленый, подключены к Ардуино, как показано на схеме:



Напишите программу, которая попеременно мигает красным и зеленым светодиодами, без "темных"интервалов. Каждый светодиод включен LED_PERIOD миллисекунд (интервал меняется от теста к тесту). Какой из светодиодов зажигается первым, роли не играет. Имеются функции pinMode(), digitalWrite(), delay().

Решение

Это еще одна супер-легкая задача, рассчитанная на то, чтобы дать вам разобраться с написанием кода для мини-симулятора. Раз диоды подключены "встречно программа должна попеременно подавать LOW на D11 и HIGH на D12 (включен LED2) или HIGH на D11 и LOW на D12 (включен LED2). Для мини-симулятора "одновременными"считаются действия, выполняемые без вызова delay() между ними.

Прежде, чем писать собственно код для переключения, не забываем инициализировать соответствующие пины как выходные. При этом обязательно использовать именованные константы, приведенные в шаблоне кода, а не числа:

```
void setup()
{
    pinMode(LED_PIN1, OUTPUT);
        pinMode(LED_PIN2, OUTPUT);
}
```

Вот самое простое (и тупое) решение для поочередного мигания (и опять-таки, обязательно используем параметр LED_PERIOD, приведенный в шаблоне кода. Он будет принимать разные значения для разных тестов, и иначе программа правильный ответ не выдаст):

```
void loop()
{
    digitalWrite(LED_PIN1, HIGH);
    digitalWrite(LED_PIN2, LOW);
    delay(LED_PERIOD);
    digitalWrite(LED_PIN1, LOW;
    digitalWrite(LED_PIN2, HIGH));
    delay(LED_PERIOD);
}
```

А вот чуть более интересное решение с использованием переменной:

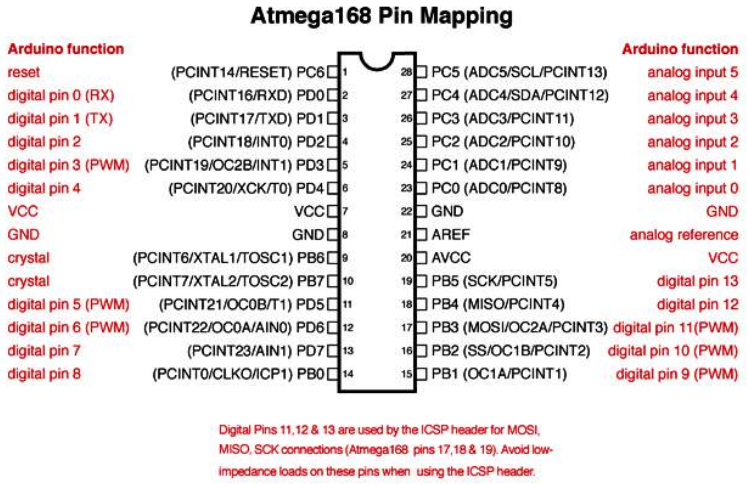
```
bool b = true;

void loop()
{
    digitalWrite(LED_PIN1, b);
    b = !b;
    digitalWrite(LED_PIN2, b);
    delay(LED_PERIOD);
}
```

Отступление (для продвинутых):

В реальном контроллере между соседними вызовами digitalWrite() будет, разумеется, некая микросекундная задержка. Для данной задачи это никакой роли не играет, но в некоторых ситуациях может быть важным. Используя только вызовы из библиотеки Ардуино, невозможно строго одновременно управлять несколькими пинами.

Сам микроконтроллер ATMEGA, тем не менее, это делать позволяет. На аппаратном уровне, "пины" (входы/выходы) контроллеры объединены в 8-битовые "порты", доступные как регистры контроллера, а на уровне языка C - как специальные переменные. Таким образом, можно строго одновременно, за 1 такт микроконтроллера, менять состояние до 8 пинов!

Если вы загуглите "ATMEGA ArduinoUNO pin mapping то найдете, например, вот такую картинку:



В контексте данной задачи, нам важен тот факт,что пинам Arduino D11 и D12 соответствуют биты 3 и 4 порта B контроллера ATMEGA. На языке C, к ним можно обратиться следующим образом (причем это действует не только в "настоящей"среде разработки AVRStudio, но и в среде Arduino!):

```
void setup()
{
   DDRB =  0b00011000;    // заменяет pinMode, устанавливая PB3 и PB4 в OUTPUT
   PORTA = 0b00010000;    // начальное состояние: PB3(D11) = 0, PB4(D12) = 1
}
void loop()
```

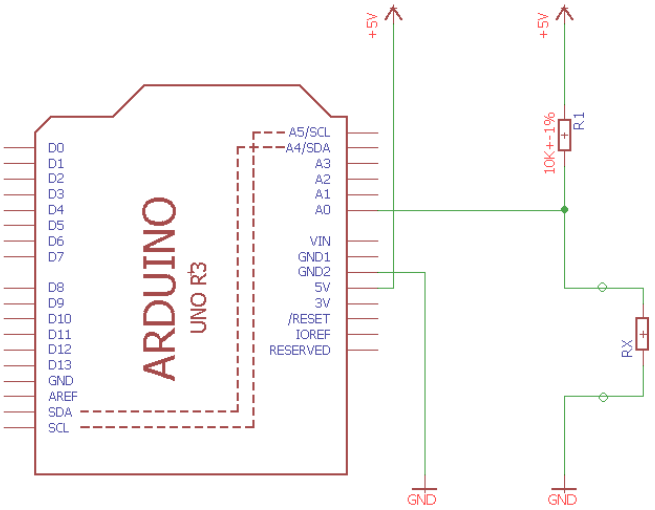
```
7  {
8      PORTA ^= 0b00011000;  // одной командой переключаем оба пина!
9      delay(LED_PERIOD);
10 }
```

Такая программа не только выполняется на реальном контроллере гораздо быстрее, но и занимает меньше места в памяти программ, чем код с вызовами функций библиотеки Ардуино. Обратите внимание, что если бы светодиоды были подключены, например, к пинам D7 и D8, то показанный выше трюк у нас бы не прошел, т.к. эти пины относятся к разным портам и не могут изменяться одной командой.

К сожалению, в мини-симуляторе низкоуровневая работа с портами не поддерживается, поэтому приведенный выше код не может быть решением данной задачи в Stepik. Тем не менее, любознательным среди вас составители задания советуют попробовать такую программку на реальном Ардуино и, по документации к контроллеру ATMega разобраться с псевдо-переменными PORTx, PINx и DDRx.

Задача 4.2.3. Простейший омметр (2 балла)

Вам нужно реализовать на Arduino простейший омметр (измеритель сопротивления), как показано на схеме.



Для этого к пину A0 подключен делитель напряжения, состоящий из прецизионного резистора 10 кОм "сверху"и измеряемого резистора "снизу". После того, как очередной измеряемый резистор подключен к схеме, контроллер включается, программа на Arduino производит измерение и через последовательный порт выводит в виде числа значение сопротивления, в кОм, с округлением до целого и с переводом строки на конце.

Для упрощения, мы принимаем, что измеряемое сопротивление всегда целое число килоом и отсутствуют любые ошибки измерения. Все значения измеряемых резисторов находятся в диапазоне от 10 кОм до 90 кОм (т.е. допускающем их сравнительно точное измерение при 10кОм подтягивающем резисторе).

Напишите программу для мини-симулятора Arduino, которая выполняет измерения.

Данные для измерения поступают из входного потока и расшифровываются мини-симулятором, а вычисленные вашей программой и отправленные на Serial.println() значения передаются в выходной поток для проверки. Мини-симулятор однократно вызывает функцию loop() для выполнения каждого измерения.

Разрешается пользоваться функциями pinMode(), analogRead() , а также упрощенным классом Serial с методами begin(), print() и println(). Передача данных должна происходить на скорости 9600 бод.

Решение

Очевидно, что это задача на использование имеющегося в микроконтроллере ATMEGA аналого-цифрового преобразователя (АЦП). На уровне библиотеки Arduino, чтение аналогового значения на входе АЦП выполняется функцией analogRead(). Эта функция возвращает значения в диапазоне от 0 (на входе - нулевое напряжение) до 1023 (напряжение питания). При подключении на аналоговый вход делителя напряжения, изображенного на схеме, напряжение на входе будет составлять:

$$V_x = V_{cc} \cdot R_x/(R_1 + R_x),$$

а значение, принятое с АЦП:

$$X_{ADC} = 1023 \cdot R_x/(R_1 + R_x)$$

Решая это уравнение относительно R_x , получаем:

$$R_x = R_1 \cdot X_{ADC}/(1023 - X_{ADC})$$

При реализации вычислений на C, всегда следует обращать внимание на правильность задания типов данных и на возможные ошибки округления. Наш код может выглядеть так:

```
#include <math.h>
Float R1 = 10.0;
void loop()
{
    int x = analogRead(A0);
    float R = R1 * x / (1023 - x);
    Serial.println(round(R));
}
```

Разумеется, нельзя забывать и про правильную инициализацию как пинов, так и последовательного порта (хотя при старте Arduino все пины уже и так находятся в режиме INPUT, желательно прописать pinMode() просто для большей ясности):

```
void setup()
{
    pinMode(A0, INPUT);
    Serial.begin(9600);
}
```

4.3. Arduino: АЦП и делители напряжения

Задача 4.3.1. Асинхронный маячок (3 балла)

К цифровым пинам контроллера Ардуино подключены 3 светодиода разных цветов (красный, желтый и зеленый). Напишите программу, которая будет одновременно мигать этими светодиодами, причем каждым - со своим периодом и скважностью. В начальный момент времени все 3 светодиода должны включиться одновременно.

Входные данные мини-симулятор считывает автоматически, но вам нужно использовать переменные, как указано в шаблоне кода. Как и в предыдущем примере, вам доступны функции pinMode(), digitalWrite() и delay().

Решение

Задачка с тремя светодиодами требует совсем другого подхода: мы должны использовать в конце функции loop() ровно одну короткую задержку (например, на 1 мс) и для каждого светодиода использовать отдельную переменную-счетчик, которая увеличивается при каждом проходе цикла. При заданных пороговых значениях счетчика светодиод может включаться или отключаться. По сравнению с фиксированной задержкой в конце цикла, время выполнения остальной части кода (проверки счетчиков и управление пинами) пренебрежимо мало.

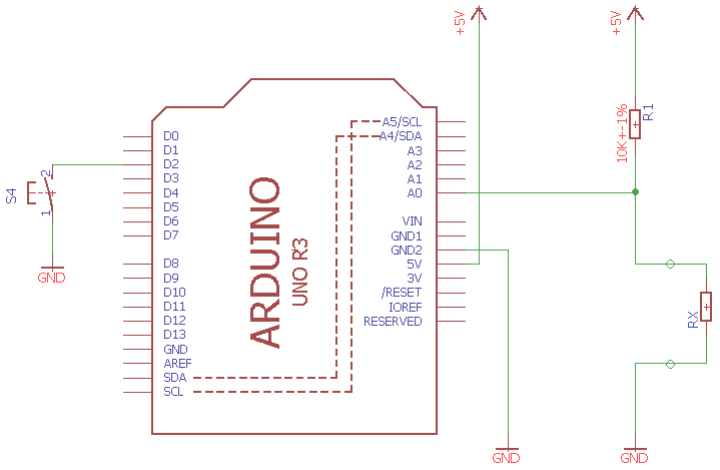
Пример программы-решения

```
int PIN_RED = 3;         // на этих пинах ваши светодиоды
int PIN_YELLOW = 4;
int PIN_GREEN = 5;

// Для каждого LED задается полный период мигания и длительность
// во включенном состояния.
// Данные считываются мини-симулятором из входного потока в эти переменные.
//и будут разными для каждого теста.

int RED_PERIOD, RED_ON_PERIOD;       // период мигания и длительность включенного
// состояния для КРАСНОГО
int YELLOW_PERIOD, YELLOW_ON_PERIOD; // то же для ЖЕЛТОГО
int GREEN_PERIOD, GREEN_ON_PERIOD;   // то же для ЗЕЛЕНОГО

int cnt_r=0, cnt_y=0, cnt_g=0; // счетчики для светодиодов

void setup()
{
    pinMode(PIN_RED, OUTPUT);
    pinMode(PIN_YELLOW, OUTPUT);
    pinMode(PIN_GREEN, OUTPUT);
}

void blink_led(int pin, int &cnt, int on_period, int period)
{
    if( !cnt ) //в начале периода ставим HIGH
        digitalWrite(pin, HIGH);
    else if( cnt >= on_period ) //по истечении времени включения - выключаем
        digitalWrite(pin, LOW);
```

```
30          cnt = (cnt+1) % period;  // циклически увеличиваем счетчик по модулю period
31      }
32
33      void loop()
34      {
35          blink_led(PIN_RED, cnt_r, RED_ON_PERIOD, RED_PERIOD);
36          blink_led(PIN_YELLOW, cnt_y, YELLOW_ON_PERIOD, YELLOW_PERIOD);
37          blink_led(PIN_GREEN, cnt_g, GREEN_ON_PERIOD, GREEN_PERIOD);
38          delay(1);
39      }
```

Задача 4.3.2. Усовершенствованный омметр (4 балла)

В задании на прошлой неделе вам предлагалось запрограммировать простой измеритель сопротивления. Вернемся вновь к этой теме, но внеся небольшие усовершенствования:



1. Теперь между пином D2 и землей подключен нормально-разомкнутый кнопочный выключатель. Измеряемые резисторы можно заменять без выключения устройства. После того, как очередной измеряемый резистор подключен к схеме, пользователь кратковременно нажимает кнопку. В момент отпускания кнопки, программа на Arduino производит измерение и выводит в виде числа значение сопротивления через последовательный порт. Если измерение проведено до отпускания кнопки, или более чем через 10ms после этого момента, analogRead() возвращает значение 1023 (как если бы измеряемый резистор еще не подключили).
2. Измеряемые сопротивления, в диапазоне от 10 кОм до 82 кОм, выбираются из номинального ряда E12 (Гугл в помощь!). Более того, как у настоящих резисторов, значения будут отклоняться от номиналов на случайную величину, не превышающую 10% номинала. Вашей программе нужно привести измеренное сопротивление к номиналу и вывести его в виде целого числа (в кОм) вызовом Serial.println(). Например, если ваша программа получила значение 54 кОм, то должно быть возвращено число 56 - ближайшее значение из номинального ряда. Данные должны передаваться на скорости 9600 бод.

Напишите программу для мини-симулятора Arduino, которая реализует описанный выше функционал: ждет нажатия и отпускания кнопки, проводит измерение, обрабатывает и выводит результат. Разрешается пользоваться функциями pinMode(),

digitalRead(), analogRead() и delay(), а также упрощенным классом Serial с методами begin(), print() и println().

ВАЖНО: в отличие от реального Arduino, в программе для мини-симулятора любые циклы ожидания должны включать вызов функции delay(), иначе в симуляторе "время не идет"и ваша программа "зациклится не сможет дождаться никаких внешних событий.

Стандартный вывод
19 818 869 518 906 523 741 510 692 556 869 517 789 509 739 844 849 614 648 500

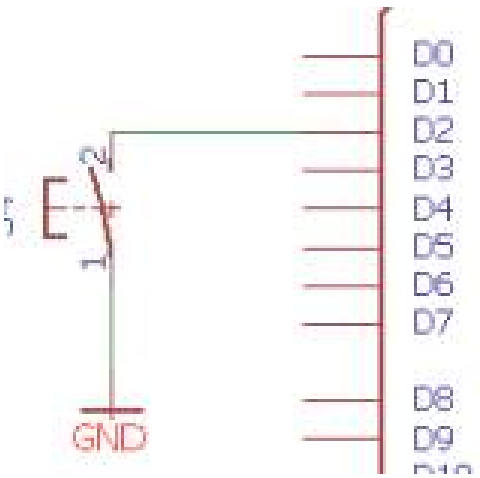
Стандартный вывод
39 56 10 82 10 27 10 22 12 56 10 33 10 27 47 47 15 18 10

Решение

Очевидно, что собственно измерение сопротивления выполняется здесь точно так же, как и в предыдущей задаче. Отличаться будет цикл ожидания нажатия кнопки до измерения и обработка ("нормализация") полученного значение после. Используя метод проектирования сверху-вниз, запишем сначала новую функцию loop(), а затем уже будем разбираться с дополнительными функциями:

```
float R1 = 10.0;
void loop()
{
    waitForButton(BUTTON_PIN);
    int x = analogRead(A0);
    float R = R1 * x / (1023 - x);
    Serial.println(normalizeResistor(R));
}
```

Чтобы поймать момент отпускания кнопки, нам сначала надо дождаться ее нажатия. Надо понимать, что при подключении кнопки между пином и землей, без внешнего подтягивающего резистора, режим пина должен быть обязательно выставлен в INPUT_PULLUP, иначе состояния пина при разомкнутой кнопке будет неопределенным. При этом, при замыкании кнопки на пине окажется 0 (LOW), а в разомкнутом состоянии - 1 (HIGH).



С учетом этого:

```
void waitForButton(int pin)
{
        while(digitalRead(pin))   // ждем нажатия кнопки
         delay(1);
        while(!digitalRead(pin))  // а затем - ее отпускания
         delay(1);
}
```

ВАЖНО: в отличие от реального Arduino, в программе для мини-симулятора любые циклы ожидания должны включать вызов функции delay(), иначе в симуляторе "время не идет"и ваша программа не сможет дождаться никаких внешних событий.

Следующее, с чем надо разобраться - это приведение измеренного значения к ближайшему стандартному значению из номинального ряда E12 (см. Википедию "Ряды номиналов радиодеталей"). Для начала, просто зададим этот ряд (в диапазоне 10 кОм - 90 кОм) в виде массива констант:

```
const int E12[] = {10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82 };
```

Самый простой способ нормализовать измеренное значение - просто сравнивать его поочередно с каждым из измеренных значений. Если отклонение не превышает $\pm 10\%$, то мы нашли нужный номинал:

```
int normalizeResistor(float R)
{
    for( int i = 0; i < 12; i++ ){
        int rn = E12[i];
        if( R >= 0.9 * rn && R < 1.1 * rn ) return rn; // номинал найден
    }
    return round(R);   // не должно случиться - возвращаем без нормализации
}
```

Такой алгоритм, однако, может давать неоднозначные ответы, поскольку 10% диапазоны для соседних значений слегка перекрываются. Например: $10 \cdot 1.1 = 11$, а $12 \cdot 0.9 = 10.8$. Таким образом, измеренное значение 10.9 кОм может соответствовать, строго говоря, номиналам как 10 кОм, так и 12 кОм. В данной задаче это не приведет к проблемам, так как генерируемые тестовые данные не будут настолько сильно отклоняться от своих номиналов, чтобы попасть в "серые зоны".

Тем не менее, чтобы избежать неоднозначности в определении номинала, можно сравнивать измеренное значение со средними арифметическими соседних номиналов. Например, для пары номиналов 12 кОм и 15 кОм естественной границей будет $(12 + 15)/2 = 13.5$ кОм. Вот как это можно сделать:

```
int normalizeResistor(float R)
{
  if( R < 0.9 * E12[0] ) return R; // меньше меньшего - возвращаем как есть
  if( R > 1.1 * E12[11] ) return R; // больше большего - возвращаем как есть

  for( int i = 0; i < 11; i++ ){    // не включая последний элемент
      float rn = (E12[i] + E12[i+1]) / 2.0;  // Граница между номиналами
      if( R < rn ) return E12[i]; // номинал найден
   }
   return E12[11];   // последний номинал
}
```

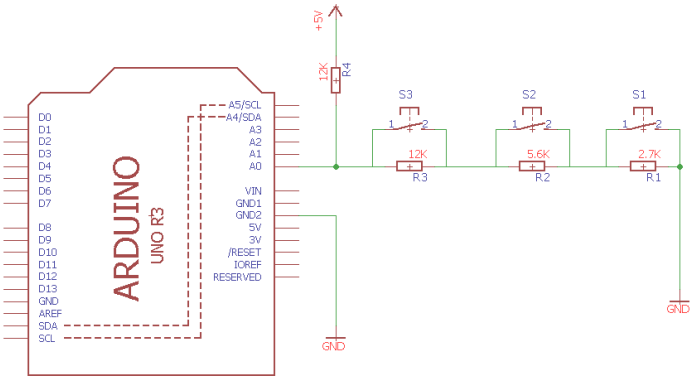
Ну и, наконец, не забываем об инициализации в функции setup(). По сравнению с предыдущей задачей, здесь добавляется установка режима INPUT_PULLUP для пина, к которому подключена кнопка:

```
void setup()
{
    pinMode(A0, INPUT);
    pinMode(BUTTON_PIN, INPUT_PULLUP);
    Serial.begin(9600);
}
```

Задача 4.3.3. Аналоговые кнопки (4 балла)

Если нужно подключить к устройству несколько кнопок, а пины контроллера приходится экономить, то подключают цепь из нескольких кнопок и резисторов к одному аналоговому входу, так что при разных комбинациях нажатий получаются разные сопротивления цепи, и, соответственно, разные напряжения на аналоговом входе.

И вот, имеется такая схема:



Напишите программу, которая сразу после включения определяет нажатые кнопки и передает через последовательный порт одно десятичное число, соответствующее их комбинации. Каждой из кнопок S1..S3 соответствует один бит в полученном числе. Например, нажатые кнопки S1 и S2 должны давать число 3 (2+1). Ожидаемая скорость передачи - 115200 бод. Программа должна учитывать, что сопротивления резисторов могут несколько (до 2.5% в каждую сторону) отклоняться от указанных на схеме значений.

В этом примере, мини-симулятор будет выполнять функцию loop() однократно для каждой комбинации нажатых кнопок. Никаких задержек и ожиданий в коде писать не надо. Используются функции analogRead(), pinMode() и упрощенный Serial, как в предыдущих заданиях.

Решение

Если нужно подключить к устройству несколько кнопок, а пины контроллера приходится экономить, то часто подключают цепь из нескольких кнопок и резисторов к одному аналоговому входу, так что при разных комбинациях нажатий получаются разные сопротивления цепи, и, соответственно, разные напряжения на аналоговом входе.

В данной схеме "верхний"резистор делителя $R4 = 12$ кОм, а "нижняя"ветка делителя состоит из 3-х последовательно соединенных резисторов ($R1 = 2.7$ К, $R2 = 5.6$ К, $R3 = 12$ К), каждый из которых может быть шунтирован при нажатии соответствующей кнопки. Для любой заданной комбинации кнопок (из 8 возможных) можно заранее вычислить напряжение на пине A0. Например, при замкнутых $S1$ и $S3$ в нижней части делителя останется только $R2 = 5.6$ К, и результат analogRead() будет равен:

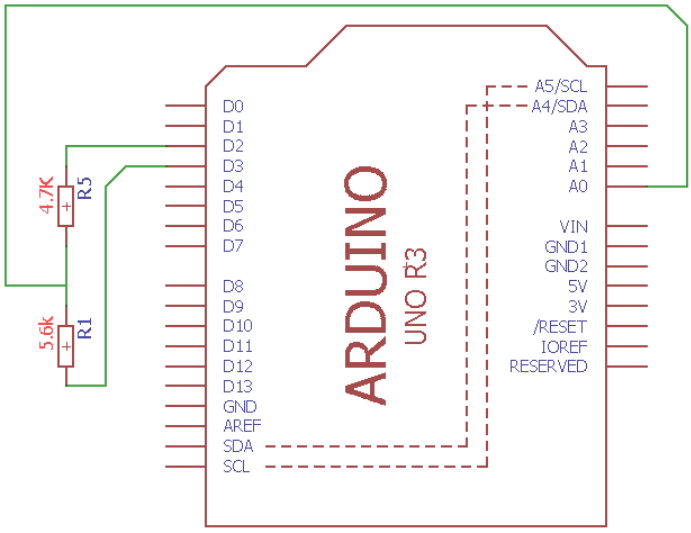
$$A = R2/(R2 + R4) \cdot 1023 = 5.6/(5.6 + 12) \cdot 1023 \approx 325$$

С учетом 2.5% допусков, диапазон значений analogRead() для этой комбинации кнопок составляет: $A_{min} = A \cdot 0.975 \approx 317$, $A_{max} = A \cdot 1.025 = 333$

Программа сравнивает полученное значение с каждым из рассчитанных таким образом диапазонов, определяя комбинацию нажатых кнопок. (Полный текст программы не приводится).

Задача 4.3.4. Загадочный делитель напряжения (1 балл)

Делитель напряжения из двух резисторов разного номинала подключен между пинами D2 и D3, а средняя точка подключена к аналоговому пину A0, как показано на схеме:



Значение напряжения измеряется при разных комбинациях настроек пинов. Считать, что напряжение на пине в выходном режиме в точности равно либо 0, либо напряжению питания (хотя в реальных схемах, логический "0"всегда чуть выше 0V, а логическая "1"чуть ниже напряжения питания). Сопротивление внутреннего подтягивающего резистора (на любом пине) принять за 10 кОм.

Нужно написать программу, которая рассчитывает все возможные напряжения, которые можно намерить на 0000000000A0, изменяя настройки пинов. В этой задаче не используется мини-симулятор Ардуино, вы просто пишете код на любом удобном для вас языке.

Формат входных данных

Два числа через пробел - сопротивления резисторов R1 и R2.

Формат выходных данных

На первой строке - число значений в ответе, на второй строке - последовательность целых чисел в диапазоне 0..1023, которые можно было бы получить функцией analogRead(A0) при различных настройках пинов.

Все числа в ответе должны быть на одной строке через пробел, упорядочены по возрастанию, повторяющиеся значения удалены.

Допускается отклонение вычисленных значений на 1 в любую сторону.

Решение

Чтобы понять эту задачу, следует помнить, что любой пин Arduino может находится в 4-х разных состояниях:

Состояние	Обозн.	Что происходит
pinMode(pin, OUTPUT); digitalWrite(pin, HIGH);	1	На пин подано напряжение питания
pinMode(pin, OUTPUT); digitalWrite(pin, LOW);	0	На пин подается 0 (подключен GND)
pinMode(pin, INPUT);	I	Пин отключен от каких-либо напряжений
pinMode(pin, INPUT_PULLUP);	U	Пин подключен к напряжению питания через подтягивающий резистор в 10K

Все 4 перечисленных режима относятся как к пинам D2, D3, так и к пину A0, на котором производятся измерения. Когда мы говорим про измерение аналогового сигнала функцией analogRead(), всегда предполагается, что аналоговый пин находится в режимах INPUT или INPUT_PULLUP. Однако ничего не мешает установить A0 в режим OUTPUT, подать на него LOW или HIGH и прочитать значение полученного напряжения, если это зачем-то вдруг понадобилось.

Итак, мы можем получить $4 \cdot 4 \cdot 4 = 64$ комбинации состояний 3х пинов. Многие из них дадут уникальные комбинации сопротивлений и уникальные значения измеряемого напряжения на средней точке делителя. Следующая функция на Python вычисляет выходное напряжение при различных комбинациях состояний трех пинов (обозначены '0', '1', 'U', 'I' - см. выше):

Пример программы-решения

Ниже представлено решение на языке Python3

```
R1 = 4.7
R2 = 5.6
R_Int = 10.0

def find_value(A, D1, D2):
  # A0 - OUTPUT, полностью определяет результат
  if A == '0':
    return 0
  elif A == '1':
    return 1023

  if (D1 in '1U') and (D2 in '1U'):  # оба HIGH or PULLUP
    return 1023

  if D1 == '0' and D2 == '0':  # R1,R2 параллельно на GND
    if A == 'I':
      return 0  #  GND независимо от R1/R2
```

```
    Else:   #  R1, R2 параллельно, R_int сверху делителя
      r_dn = 1 / (1/R1 + 1/R2)
      r_up = R_Int
      return int(1023 * r_dn / (r_up + r_dn) )

  # Считаем верх делителя: начать с бесконечности, учитывать все
  # резисторы, идущие параллельно
  r_up = math.inf
  if A == 'U':
    r_up = 1 / (1/R_Int + 1/r_up)
  if D1 == '1':
    r_up = 1 / (1/R1 + 1/r_up)
  elif D1 == 'U':  # R_Int встает последов. с R1
    r_up = 1 / (1/(R1+R_Int) + 1/r_up)

  if D2 == '1':
    r_up = 1 / (1/R2 + 1/r_up)
  elif D2 == 'U':
    r_up = 1 / (1/(R2+R_Int) + 1/r_up)


  # Аналогично с "нижними" резисторами, но тут только могут быть R1 и R2
  r_dn = math.inf
  if D1 == '0':
    r_dn = 1 / (1/R1 + 1/r_dn)
  if D2 == '0':
    r_dn = 1 / (1/R2 + 1/r_dn)


  # Все 3 пина поставлены на INPUT, неопределенное напряжение.
  if r_up == math.inf and r_dn == math.inf:
    return None

  if r_dn == math.inf:
    return 1023
  if r_up == math.inf:
    return 0

  return int(1023 * r_dn / (r_up + r_dn) )


# Полный перебор состояний пинов
results = []
for A0 in "IU01":
  for D1 in "IU01":
      for D2 in "IU01":
        v = find_value(A0, D1, D2)
        if not (v is None):
          results.append(v)


# Печатаем все уникальные решения, с сортировкой по возрастанию.
arr = [str(v) for v in sorted(set(results))]
print(len(arr))
print(' '.join(arr))
```

4.4. Шаговые двигатели и координатные устройства

Шаговые двигатели - основа самых разных координатных устройств, в том числе станков с ЧПУ, а среди них и самых распространенных - 3D-принтеров (да и обычных бумажных принтеров тоже!)

В отличие об обычных электромоторов, шаговый двигатель не просто "вращается а поворачивается на заданный угол, с весьма хорошей точностью, при подаче импульсов определенной формы, полярности и продолжительности на его 2 обмотки. Для управления шаговыми двигателями используют специальные ' микросхемы-драйверы. Типичный драйвер управляется с микроконтроллера по 3-м пинам ENABLE, DIR и STEP:

ENABLE - включение двигателя. Когда шаговый двигатель включен, ток протекает по обмоткам и блокирует движение ротора, так что прокрутить вал можно, только приложив значительное усилие. Часто сигнал ENABLE для драйвера имеет инверсную полярность, т.е. двигатель включается при установке низкого уровня сигнала.

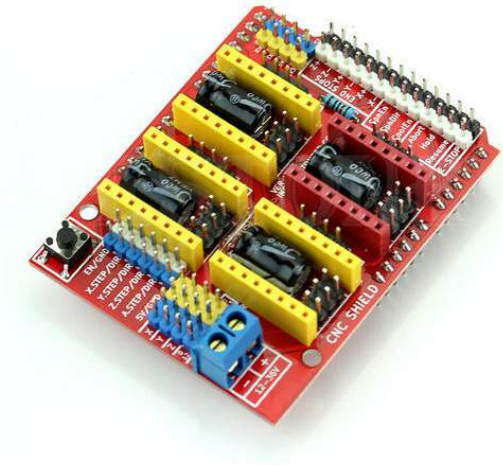
DIR - уровень сигнала определяет направление вращения.

STEP - каждый короткий импульс, подаваемый на этот пин, поворачивает вал шагового двигателя на 1 шаг.

Драйвер шагового двигателя несложно подключить к Ардуино прямо на макетной плате, но гораздо удобнее пользоваться специальными шилдами. На следующей картинке показан"бутерброд"из Arduino UNO (внизу), CNC ShieldV3 и 4-х драйверов моторов сверху. Это типичный расклад для любительского настольного фрезерного станка или плоттера. Для 3D-принтеров чаще применяют аналогичный "бутерброд но на базе Arduino Mega и шилда RAMPS.

Задача 4.4.1. Какие пины? ("Google it!") (1 балл)

Когда мы подключаем драйвер шаговых двигателей на макетной плате, то мы свободны в выборе пинов Arduino, которые будут им управлять. Но если мы используем CNC Shield, то вся "распиновка"определяется платой. Найдите в интернете информацию о CNC Shield V3 для Arduino UNO и заполните пробелы.



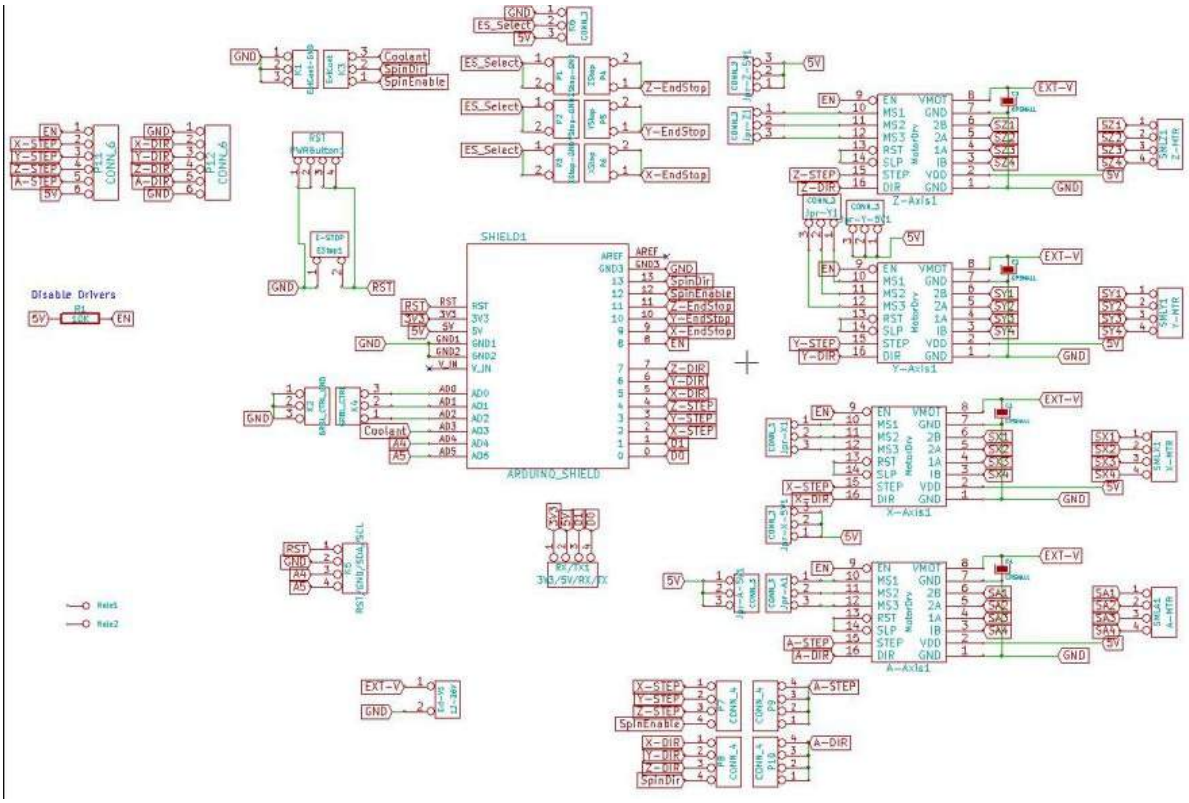
а) При использовании этой платой сигнал ENABLE:
   (a) Отдельный для каждого мотора
   (b) Общий для всех моторов
   (c) Всегда включен
   (d) Общий для каналов X и Y

б) Сигнал STEP для канала X подключен к пину Arduino:
   (a) D1
   (b) D2
   (c) D3
   (d) D4
   (e) D5
   (f) D6
   (g) D7
   (h) D8

в) Сигнал DIR для канала Z подключен к пину Arduino:
   (a) D1
   (b) D2
   (c) D3
   (d) D4
   (e) D5
   (f) D6
   (g) D7
   (h) D8
   (i) D9
   (j) D10

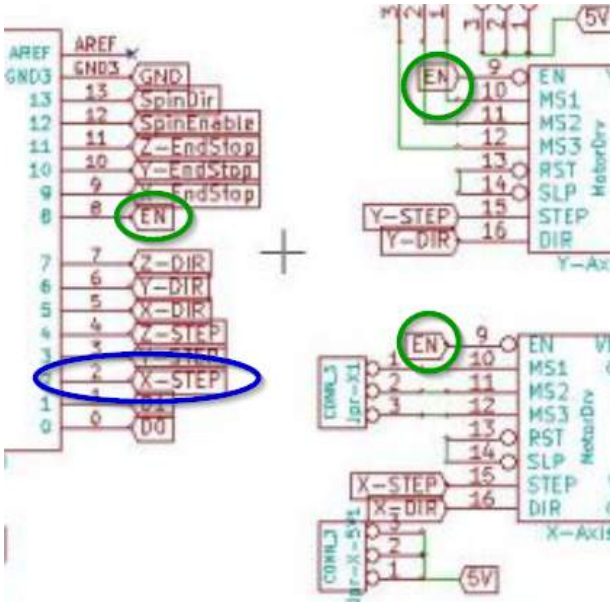
Решение

Решение этой задачи очевидно из условия. Ищем "CNC shield pinout", и находим вот такую схему:



Посмотрев подробнее на подключение драйверов, мы видим, что:

- Сигнал EN (а точнее - NOT ENABLE, т.к. драйверы включаются при подаче 0 на EN) разведен с пина 8 на все драйвера,
- Сигнал шаг X_STEP подключен к пину 2, Y_STEP к пину 3, Z_STEP к пину 4
- Сигнал направления X_DIR подключен к пину 5 и т.д.



Этого достаточно, чтобы правильно ответить на вопрос задания. На практике, следует иметь в виду, что разводка конкретной "китайской" платы может отличаться от найденной вами в интернете, поэтому крайне желательно проверять ее по дорожкам на плате и/или мультиметром.