MATH-GA 2840 HW#5

Yifei(Fahy) Gao yg1753

1. (Fourier coefficients and smoothness) Let $x : \mathbb{R} \to \mathbb{C}$ be periodic with period 1 and let $\hat{x}[k]$ denote the k th Fourier coefficient of x, for $k \in \mathbb{Z}$ (computed on any interval of length 1)

(a) Suppose x is continuously differentiable. Prove that for $k \neq 0$ we have

$$|\hat{x}[k]| \leq \frac{C_1}{|k|}$$

for some $C_1 \geq 0$ that depends on x (but not on k). [Hint: Integration by parts. Also note that

$$\left|\int_0^1 f(t)dt\right| \leq \int_0^1 |f(t)|dt < \infty$$

if f is continuous on [0,1] .]

Since $T = 1, a = 0$ then based on the formula of Fourier series:

$$|\hat{x}[k]| = \left|\int_0^1 x(t)\exp(-i2\pi kt)dt\right|$$

$$= \left|-\frac{x(t)}{i2\pi k}\exp(-i2\pi kt)\right|_0^1 + \int_0^1 \frac{x'(t)}{i2\pi k}\exp(-i2\pi kt)dt|$$

$$= \left|\int_0^1 \frac{x'(t)}{i2\pi k}\exp(-i2\pi kt)dt\right| \text{ (} x \text{ has the period of 1)}$$

$$(\left|\int_0^1 f(t)dt\right| \leq \int_0^1 |f(t)|dt) \leq \int_0^1 \left|\frac{x'(t)}{i2\pi k}\exp(-i2\pi kt)\right| dt$$

$$= \left|\frac{1}{k}\right| \int_0^1 \left|\frac{x'(t)}{i2\pi}\right| dt(\text{, since } \exp(-i2\pi kt) = cos(2\pi k) - isin(2\pi k) = 1)$$

$$= \frac{C_1}{|k|} \blacksquare$$

(b) Suppose x is twice continuously differentiable. Prove that for $k \neq 0$ we have

$$|\hat{x}[k]| \leq \frac{C_2}{|k|^2}$$

for some $C_2 \geq 0$ that depends on x (but not on k).

Since x is twice continously differentiable, then most of the steps are the same as the part a, except $\left|\int_0^1 \frac{x'(t)}{i2\pi k}\exp(-i2\pi kt)dt\right| = \left|-\frac{x'(t)}{(2\pi)^2 k^2}\exp(-i2\pi kt)\right|_0^1 + \int_0^1 \frac{x''(t)}{(2\pi)^2 k^2}\exp(-i2\pi kt)dt|$, then:

$$
\begin{aligned}
|\hat{x}[k]| &= \left|-\frac{x'(t)}{(2\pi)^2k^2}\exp(-i2\pi kt)\right|_0^1 + \int_0^1 \frac{x''(t)}{(2\pi)^2k^2}\exp(-i2\pi kt)dt| \\
&= \left|\int_0^1 \frac{x''(t)}{(2\pi)^2k^2}\exp(-i2\pi kt)dt\right| \\
&\leq \left|\frac{1}{k^2}\right|\int_0^1 \left|\frac{x''(t)}{(2\pi)^2}\right|dt \\
&= \frac{C_2}{|k|^2} \blacksquare
\end{aligned}
$$

2. (Sampling a sum of sinusoids) We are interested in a signal x belonging to the unit interval [0,1] of the form
$$x(t) := a_1 \exp(i2\pi k_1 t) + a_2 \exp(i2\pi k_2 t)$$
where the amplitudes a_1 and a_2 are complex numbers, and the frequencies k_1 and k_2 are known integers. We sample the signal at N equispaced locations $0, 1/N, 2/N, \dots,$ $(N-1)/N,$ for some positive integer N

(a) What value of N is required by the Sampling Theorem to guarantee that we can reconstruct x from the samples?

By Nyquist-Shannon-Kotelnikov sampling theorem,

with cut-off frequency $\frac{k_c}{T}$, a bandlimited signal $x \in L_2[0,T)$, when $T > 0$ as long as:

$$N \geq 2k_c + 1$$

And $2k_c + 1$ is Nyquist rate where:
$$k_c = max(|k1|, |k2|)$$

Thus,
$$N \geq 2(max(|k1|, |k2|)) + 1$$

(b) Write a system of equations in matrix form mapping the amplitudes a_1 and a_2 to the samples x_N

Based on the complex sinusoid formula we have for this question, we can change the form in terms of N:
$$x(t) := a_1 \exp\left(\frac{i2\pi k_1 i}{N}\right) + a_2 \exp\left(\frac{i2\pi k_2 i}{N}\right) \text{...where } 0 \leq i \leq N-1$$

Therefore, we can write the matrix equation of $x = \psi_{k_j} A$ where $j = [1, 2]$:

$$
\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} =
\begin{bmatrix}
\exp(i2\pi k_1 0) & \exp(i2\pi k_2 0) \\
\exp\left(\frac{i2\pi k_1}{N}\right) & \exp\left(\frac{i2\pi k_2}{N}\right) \\
\vdots & \vdots \\
\exp\left(\frac{i2\pi k_1 N - i2\pi k_1)}{N}\right) & \exp\left(\frac{i2\pi k_2 N - i2\pi k_2)}{N}\right)
\end{bmatrix}
\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}
$$

(c) Under what condition on N, k_1 and k_2 can we recover the amplitudes from the samples by solving the system of equations? Can N be smaller than the value dictated by the Sampling Theorem? If yes, give an example. If not, explain why.

In order to recover the amplitudes from the sample, the ψ has to be invertible which means ψ_{k1} and ψ_{k2} have to be linearly independent that

$$k_2 - k_1 \mod N \neq 0$$

Therefore, N can be smaller than the value dictated by the Sampling Theorem becase N can be 2 at the minimum:

Let $k_1 = 1, k_2 = 2$, then N = 2 and:

$$\begin{bmatrix} x(0) \\ x(\frac{1}{N} = \frac{1}{2}) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \exp(i\pi) & \exp(i2\pi) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

(d) What is the limitation of this approach, which could make it unrealistic?

The limitation would be the the exact expression of the signal x, since in the real world there is not a realistic thing to find out that x is the summation of N sinusoids of known frequencies, and it is really hard to achieve.

3. (Sampling theorem for bandpass signals) Bandpass signals are signals that have nonzero Fourier coefficients only in a fixed band of the frequency domain. We are interested in sampling a bandpass signal x belonging to the unit interval [0,1] that has nonzero Fourierseries coefficients between k_1 and k_2, inclusive, where k_1 and k_2 are known positive integers such that $k_2 > k_1$

(a) We sample the signal at N equispaced locations $0, 1/N, 2/N, \dots, (N-1)/N$. What value of N is required by the Sampling Theorem to guarantee that we can reconstruct x from the samples?

Since $k_2 > k_1$ and based on the sampling theorem I mentioned on question 2 part a, then:

$$N \geq 2(max(|k1|, |k2|)) + 1 = 2k_2 + 1$$

(b) Assume that $k_2 := k_1 + 2\tilde{k}_c$, where $\tilde{k}_c$ is a positive integer. For any $N \geq 2\tilde{k}_c + 1$ it is possible to recover the signal from the samples. Explain why (you don't need to derive any explicit expressions).

Because the frequencies k_1 and k_2 that contruct the ψ_{k_1} & ψ_{k_2} are within the range of $2\tilde{k}_c$, which is smaller than N due to $N \geq 2\tilde{k}_c + 1$. As a result, it is positive to recover the signal.

(c) Assume that $k_2 := k_1 + 2\tilde{k}_c, N \geq 2\tilde{k}_c + 1$, and $mN = k_1 + \tilde{k}_c$ for some integer m. Explain precisely how to recover x from the samples in this case.

Since it is on a uniform grid then we can set ψ_k where $k \in [-\tilde{k}_c, \tilde{k}_c]$, then:

$$\psi_{-\tilde{k}_c} = \psi_{-\tilde{k}_c+mN} = \psi_{-\tilde{k}_c+k_1+\tilde{k}_c} = \psi_{k_1}$$

And,

$$\psi_{\tilde{k}_c} = \psi_{\tilde{k}_c+mN} = \psi_{\tilde{k}_c+k_1+\tilde{k}_c} = \psi_{k_2}, \text{ since } k_2 = k_1 + 2\tilde{k}_c$$

Then the Fourier coefficients $\hat{x}$ are recovered as:

$$\hat{x}_k = \frac{T}{N}\langle x_{[N]}, \psi_k \rangle$$

By the definition of Aliasing on the notes 2.5:

$$\hat{x}^{\text{rec}}[k] = \sum_{\{(m-k) \bmod N=0\}} \hat{x}[m]$$

then since $k_2 - k_c = k_1 + k_c = mN$, and $k_1 - (-k_c) = k_1 + k_c = mN$ where $mN \mod N = 0$, then:

$$\hat{x}^{\text{rec}}[-k_c] = \hat{x}[k_1]$$
$$\ldots$$
$$\hat{x}^{\text{rec}}[k_c] = \hat{x}[k_2]$$

Thus, we can say that

$$x = \sum_{k=k1}^{k_2} \hat{x}[k]\psi_k \blacksquare$$

4. (Frequency analysis of musical notes) In this exercise you will use the code and data in the musicdata folder. Make sure you have the python packages sklearn, pandas, sounddevice, and soundfile installed. The skeleton code for you to work with is given in analysis. py which uses tools given in music_tools.py. The data used here comes from the NSynth dataset.

(a) Plot the audio signals for the first signal in the training set, and the first vocal signal in the training set (i.e., the first signal whose instrument_family_str field is 'vocal' in the dataframe). In the titles of your two plots, include the instrument_family_str and the frequency (in Hz). We recommend you also use play_signal to hear what the signals sound like.

In [8]:
```
pip install sounddevice
```
...

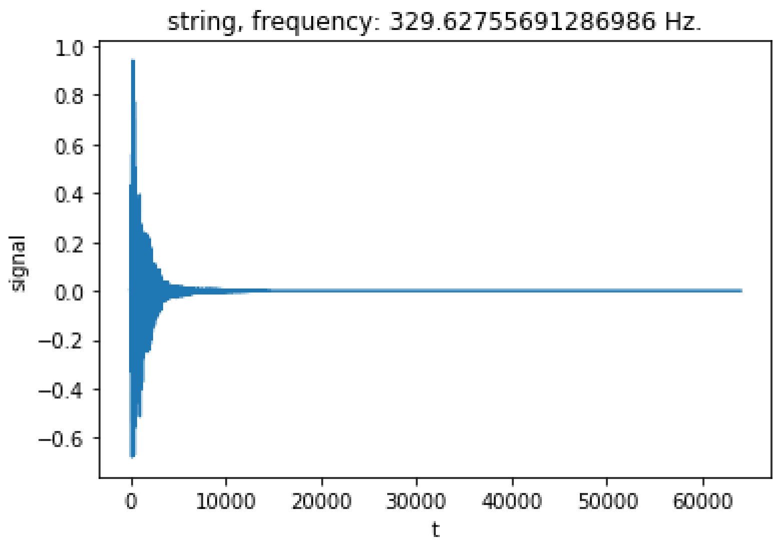
In [10]:
```
pip install pysoundfile
```
...

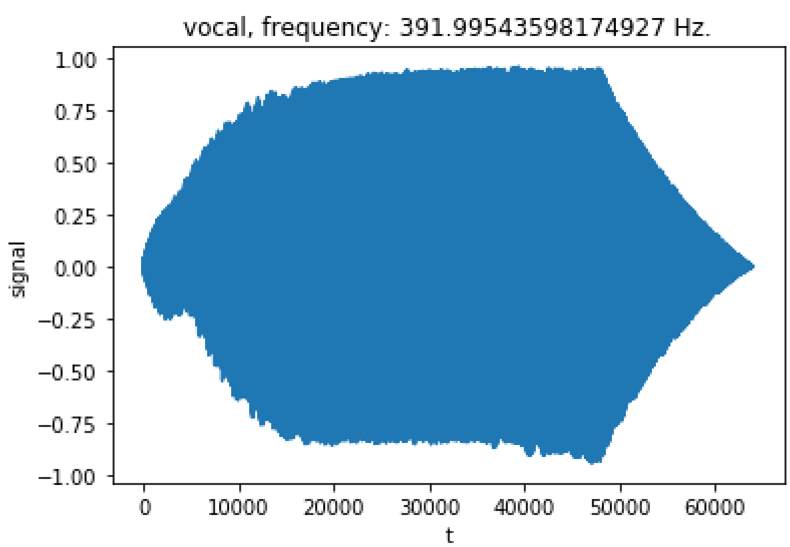
In [2]:
```
from music_tools import *
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression



df_train,df_test = load_df()
print(df_train.head())
print("Number of training examples: %d"%len(df_train))
print("Number of test examples: %d"%len(df_test))
sigs_train,sigs_test = load_signals(df_train),load_signals(df_test)
y_train,y_test = df_train['pitch'].values,df_test['pitch'].values
all_pitches = sorted({p for p in y_train})


print('Pitches:',all_pitches)
print({s for s in df_train['instrument_family_str']})

#question 1

stren_train = df_train['instrument_family_str'][0]
freq_train = df_train['frequency'][0]


plt.figure()
plt.plot(np.linspace(1,len(sigs_train[0]),len(sigs_train[0])),sigs_train[0]
plt.xlabel('t')
plt.ylabel('signal')
plt.title(stren_train + ", frequency: "+ str(freq_train)+" Hz.")

ind_first_vocal = df_train['instrument_family_str'][df_train['instrument_fa

stren_vocal_train = df_train['instrument_family_str'][ind_first_vocal]
freq_vocal_train = df_train['frequency'][ind_first_vocal]

plt.figure()
plt.plot(np.linspace(1,len(sigs_train[ind_first_vocal]),len(sigs_train[ind_
plt.xlabel('t')
plt.ylabel('signal')
plt.title( stren_vocal_train+", frequency: "+str(freq_vocal_train) +" Hz.")
```

```
                          filename   frequency  instrument_family  \
0      string_acoustic_014-064-127  329.627557                  8
1  keyboard_electronic_001-065-127  349.228231                  4
2       bass_synthetic_034-065-127  349.228231                  0
3      guitar_acoustic_010-064-100  329.627557                  3
4  keyboard_electronic_001-060-075  261.625565                  4

  instrument_family_str  pitch  sample_rate
0                string     64        16000
1              keyboard     65        16000
2                  bass     65        16000
```

```
3                  guitar     64         16000
4                keyboard     60         16000
Number of training examples: 1000
Number of test examples: 283
Pitches: [60, 62, 64, 65, 67, 69, 71, 72]
{'keyboard', 'flute', 'reed', 'brass', 'guitar', 'mallet', 'vocal', 'bas
s', 'string', 'organ'}
```

Out[2]:
```
Text(0.5, 1.0, 'vocal, frequency: 391.99543598174927 Hz.')
```





(b) For each signal in the test set, compute the (strictly positive) frequency with the largest amplitude (in absolute value), and convert it to a pitch number (using the tools in music_tools). This will be our predicted pitch.

```
i. Report what overall fraction of the signals in the test set you
 accurately predict using this method (i.e., your overall accurac
y).
```

In [3]:
```
df_test.head()
```

Out[3]:

	filename	frequency	instrument_family	instrument_family_str	pitch	sample_ra
1000	guitar_electronic_022-071-025	493.883301	3	guitar	71	160(
1001	keyboard_electronic_078-071-025	493.883301	4	keyboard	71	160(
1002	reed_acoustic_018-071-025	493.883301	7	reed	71	160(
1003	string_acoustic_014-064-025	329.627557	8	string	64	160(
1004	string_acoustic_057-072-075	523.251131	8	string	72	160(

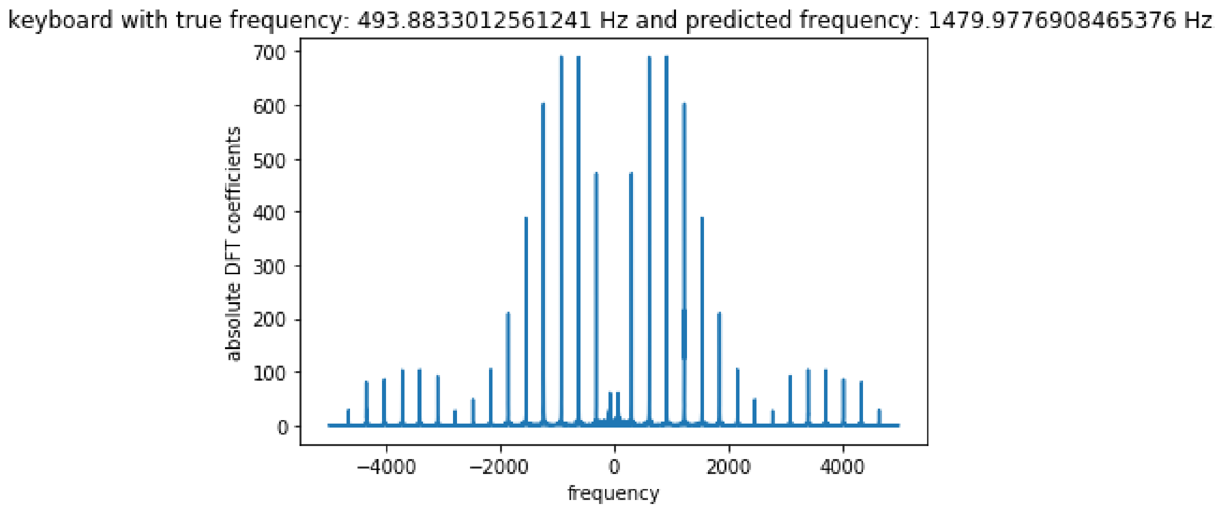
In [4]:
```
np.fft.fft?
```

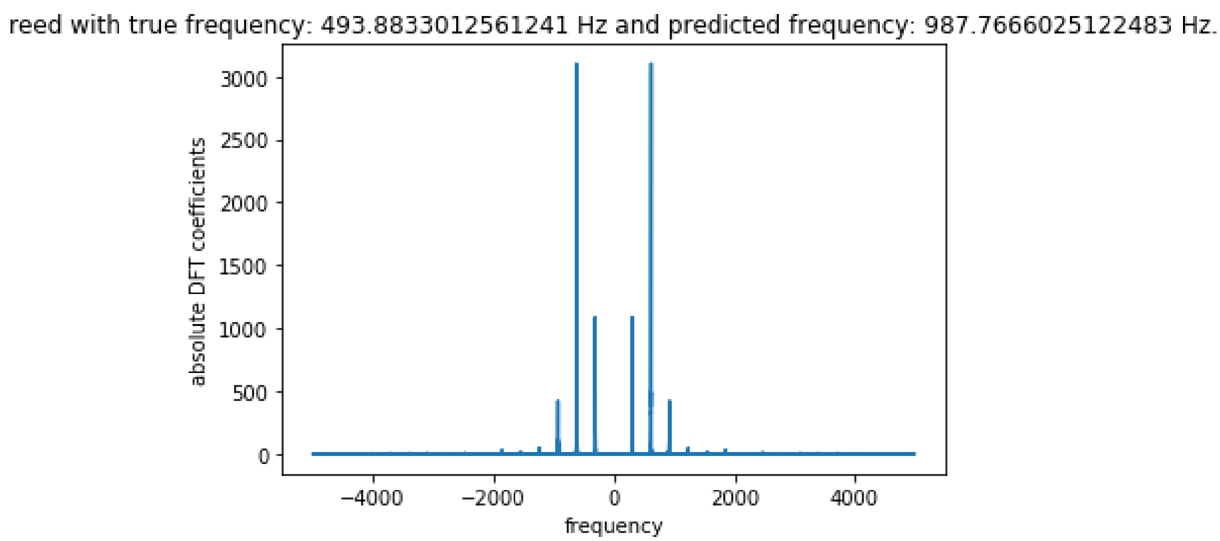
In [12]:
```
pitch_test = []
mis_fft = []
mis_fftfreq = []
mis_ind = []

for i in range(0,len(sigs_test)):
    fft_i = np.fft.fft(sigs_test[i])
    fft_i_freq = np.fft.fftfreq(len(sigs_test[i]))
    fft_i_max_ind = np.argmax(np.abs(fft_i))
    fft_i_max_freq = fft_i_freq[fft_i_max_ind]
    freq_in_hertz = abs(fft_i_max_freq * 16000)
    if freq_in_hertz == 0:
        fft_i_pitch = 0
        pitch_test.append(fft_i_pitch)
    else:
        fft_i_pitch = freq2pitch(freq_in_hertz)
        pitch_test.append(fft_i_pitch)
    if fft_i_pitch != df_test['pitch'].to_numpy()[i]:
        if len(mis_fft) < 2:
            mis_fft.append(fft_i)
            mis_fftfreq.append(fft_i_freq)
            mis_ind.append(i)
test_accuarcy = np.sum(df_test['pitch'] == pitch_test) /len(df_test['pitch'

print("The test accuracy is "+ str(test_accuarcy))
```

```
The test accuracy is 0.7208480565371025
```

```
ii. For the first two signals you misclassify (in the order they oc
cur in the test set), give plots of their absolute DFT coefficients
(use np.fft.fft and make one plot per signal). In the title of your
plots, include the instrument_family_str, the true frequency, and t
he predicted frequency (in Hz). Make sure to plot the coefficients
 on an axis centered at 0 by using fftfreq with the correct argumen
ts.
```

In [39]:
```
for i in range(0,len(mis_fft)):
    plt.figure()
    plt.plot(mis_fftfreq[i]*10000,np.abs(mis_fft[i]))
    plt.xlabel("frequency")
    plt.ylabel("absolute DFT coefficients")
    df_test_temp = df_test.reset_index()
    instru_str = df_test_temp['instrument_family_str'][mis_ind[i]]
    true_freq_test = df_test_temp['frequency'][mis_ind[i]]
    pred_freq_test = pitch2freq(pitch_test[mis_ind[i]])
    plt.title(instru_str +" with true frequency: "+str(true_freq_test) +" H
```





iii. What is the instrument family for which the method got the highest fraction of incorrect predictions (i.e., number incorrect divided by number of examples from that family)?

In [23]:
```
df_series = np.where(df_test_temp['pitch'] == pitch_test)
#df_series
```

Out[23]:
```
(array([  0,   3,   4,   6,   7,   8,  10,  12,  13,  14,  15,  16,  17,
         19,  20,  21,  24,  25,  27,  28,  29,  30,  32,  33,  34,  35,
         36,  39,  40,  41,  42,  43,  44,  45,  47,  48,  49,  50,  52,
         55,  56,  57,  58,  60,  61,  62,  63,  64,  65,  66,  67,  68,
         69,  73,  74,  75,  77,  78,  80,  81,  82,  83,  84,  85,  86,
         88,  90,  92,  93,  94,  95,  96,  97,  99, 100, 101, 102, 103,
        104, 105, 106, 108, 112, 115, 116, 117, 118, 119, 121, 122, 125,
        126, 127, 128, 129, 130, 131, 132, 134, 135, 137, 138, 139, 140,
        141, 143, 144, 145, 146, 147, 148, 149, 150, 152, 154, 157, 158,
        160, 161, 162, 163, 164, 166, 167, 169, 170, 171, 172, 173, 175,
        178, 180, 182, 184, 185, 188, 190, 191, 194, 196, 197, 200, 201,
        203, 204, 205, 206, 207, 208, 209, 211, 212, 213, 214, 215, 217,
        220, 221, 222, 223, 225, 226, 227, 228, 230, 231, 232, 233, 234,
        235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247,
        248, 250, 251, 252, 253, 254, 255, 257, 258, 259, 260, 261, 262,
        265, 268, 270, 271, 272, 274, 276, 277, 279]),)
```

In [30]:
```
df_series = df_test_temp['pitch'] == pitch_test
df_series_misclassify = ~df_series
df_series_misclassify.astype(int)
df_test_temp['misclassify'] = df_series_misclassify

total_per_str = np.unique(df_test['instrument_family_str'],return_counts=Tr
misclass_per_str =df_test_temp.groupby('instrument_family_str')['misclassif
ratio_misclassify = misclass_per_str / total_per_str[1]

ind_most_misclassify = np.argmax(ratio_misclassify.to_numpy())
str_most_misclassify = total_per_str[0][ind_most_misclassify]
print("The instrument family: "+str_most_misclassify+", which the method go
```

```
The instrument family: vocal, which the method got the highest fraction o
f incorrect predictions
```

```
iv. Why does your answer in the previous part make sense?
```

Because human voices are complex and varied by different people which is very hard to hit on the specific consistent frequency and pitch.

(c) Use the LogisticRegression class in sklearn to fit a pitch classifier on the training set using the absolute DFT coefficients as the features. Use the default parameters but set multi_class to 'multinomial' and solver to 'lbfgs'. Note: We will use the negative frequencies as well for convenience, even though they have the same magnitudes as the positive (the L_2 regularization will take care of it for us).

i. Report your score on the test set as computed by the model.

In [31]:
```
feature_DFT = np.abs(np.fft.fft(sigs_train))
feature_DFT_test = np.abs(np.fft.fft(sigs_test))
model = LogisticRegression(multi_class='multinomial',solver='lbfgs')
model.fit(feature_DFT,df_train['pitch'])
y_pred_pitch = model.predict(feature_DFT_test)
test_accuracy = np.sum(y_pred_pitch == df_test['pitch']) / len(df_test)

print("The testing accuracy is "+ str(test_accuracy))
```

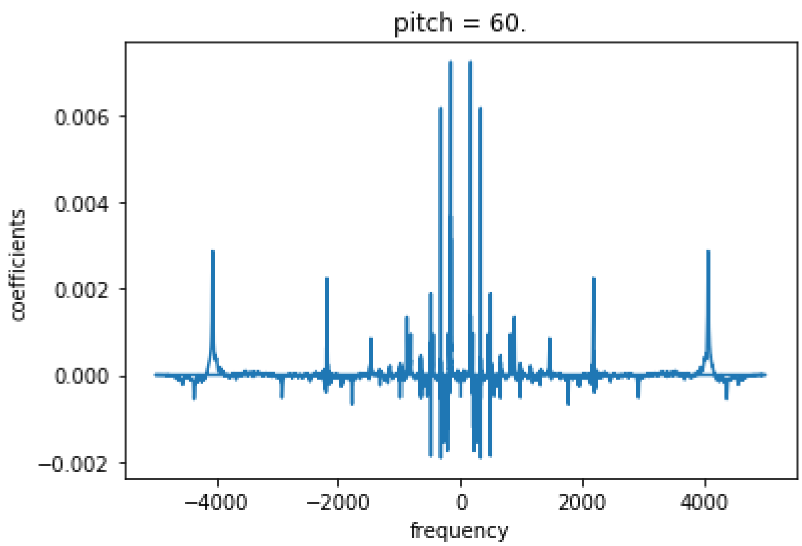
```
The testing accuracy is 0.9964664310954063.
```

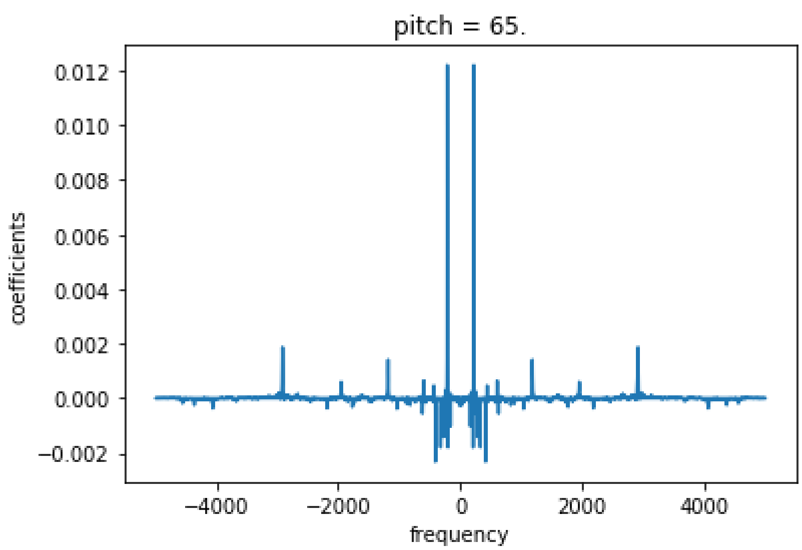
```
/Users/herculesgao/opt/anaconda3/lib/python3.7/site-packages/sklearn/line
ar_model/logistic.py:947: ConvergenceWarning: lbfgs failed to converge. I
ncrease the number of iterations.
  "of iterations.", ConvergenceWarning)
```

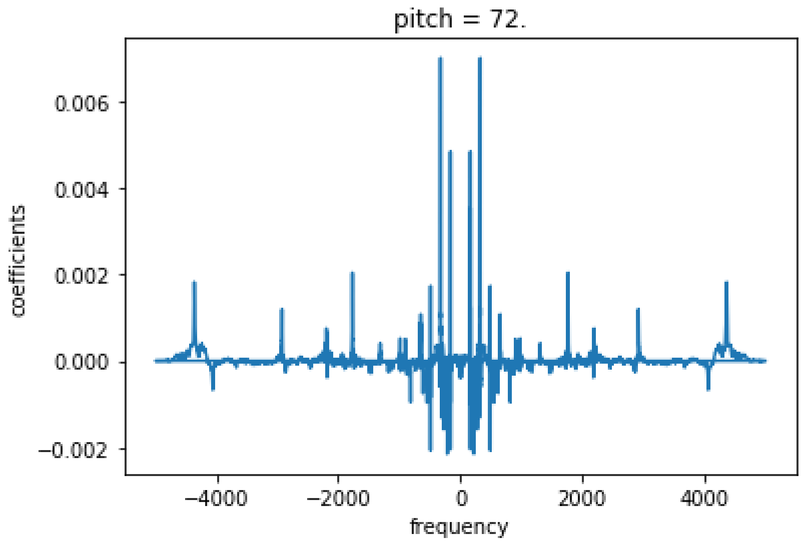
ii. Give 3 plots of the model coefficients for pitches $60, 65,$ and 72. Make sure to plot the coefficients on an axis centered at 0 by using fftfreq with the correct arguments (because the coefficients correspond to frequencies).

In [41]:

```
model_coefs = model.coef_
select_pitches=[60,65,72]
pitch_ind_selected = np.unique(df_train['pitch']).searchsorted(select_pitch
coefs_pitch_selected = model_coefs[pitch_ind_selected]
for i in range(0,len(coefs_pitch_selected)):
    plt.figure()
    plt.plot(np.fft.fftfreq(len(sigs_test[0]))*10000,coefs_pitch_selected.r
    plt.xlabel('frequency')
    plt.ylabel('coefficients')
    plt.title(f"pitch = {select_pitches[i]}.")
```







iii. Can you (very roughly) interpret the graphs in the previous part?

```
1) For the graph of pitch 60, when the frequency closes to 0 the Fo
urier coefficients rise the highest point which is higher than 0.00
6 in general, and the Fourier coefficients rise at around 4000/-400
0 frequency and around 2000/-2000 frequency.
2) For the graph of pitch 65, the Fourier coefficients is less spre
ad and only rises to 0.012 when frequency come down to around 0. Th
ere are only small bumps at around 3000/-3000 frequency and around
 1000/-1000 frequency.
3) For the graph of pitch 72, the Fourier coefficients pattern is v
ery similar to pitch 60, however the coefficients decrease a bit wh
en the frequency is really close to 0 and it has more bumps from -5
000 to 5000 frequency.
```