Calculating frequencies of signal

Analysis steps:

- load signal from file
- get sample rate and array of values
- filter signal with lowpass filter to remove high frequencies
- divide signal into chunks
- for every chunk calculate FFT - Fast Fourier Transform
- find maximum value in spectrum
- change it to frequency multiplying it by (fs / chunk_size)
- calculate MSE for frequency array
- transform MSE into user score

VoiceAnalyzer class description

Methods:

- read_signal - reads signal from file
- butter_lowpass - calculate filter params
- butter_lowpass_filter - filters signal with those params
- divide_signal_into_chunks - divides signal into array of chunks
- get_frequencies - calculate frequencies in signal
- trim_frequencies - trims frequencies array to the same length

In [19]:

```
from scipy.io import wavfile
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import butter, lfilter, freqz
from scipy.signal import stft
from scipy.fft import fft

class VoiceAnalyzer:
    def read_signal(self, path):
        fs, y = wavfile.read(path)
        return fs, y

    def butter_lowpass(self, cutoff, fs, order=5):
        nyq = 0.5 * fs
        normal_cutoff = cutoff / nyq
        b, a = butter(order, normal_cutoff, btype='low', analog=False)
        return b, a

    def butter_lowpass_filter(self, data, cutoff, fs, order=5):
        b, a = self.butter_lowpass(cutoff, fs, order=order)
        y = lfilter(b, a, data)
        return y

    def divide_signal_into_chunks(self, y, chunk_size):
        return [y[i: i + chunk_size] for i in range(0, len(y), chunk_size)]

    def get_frequencies(self, y, fs, chunk_size):
        freqs = []
        for y_fourier in y:
            y_fourier = fft(y_fourier)
```

```
            y_fourier = y_fourier[0:int(chunk_size / 2)]
            # index of max -> freq, * fs / step to get proper values
            freq = np.argmax(abs(y_fourier)) * (fs / chunk_size)
            # round to 2 decimal places
            freq = round(freq, 2)
            # change freq == 0 to 1, beacause of logarithming later
            if(freq == 0):
                freq = 1
            freqs.append(freq)

        return freqs

    def trim_frequencies(self, freq, freq2):
        if len(freq) > len(freq2):
            freq = freq[0:len(freq2)]
        else:
            freq2 = freq2[0:len(freq)]
        return freq, freq2
```

VoiceScoreCalculator class

Methods:

- set_freq, set_freq2 - frequencies arrays setters
- get_mse - calculatre MSE from 2 signals, uses VoiceAnalyzer class
- get_frequencies - returns frequencies arrays
- process_mse - apply function to MSE, covert big values to small and small to big (quadratic function) y = (x - 50000 / 1000) ^ 2
- get_score - function that apply converting methods, also returns 10^6 score for 0 MSE (if any user can achieve such result), and 0 if MSE is bigger than 50 000 (because of poor vocal performance)

In [20]:

```
class VoiceScoreCalculator:
    def __init__(self):
        self.CHUNK_SIZE = 4410
        self.LOW_CUT_FREQ = 600
        self.analyzer = VoiceAnalyzer()
        self.freq = []
        self.freq2 = []

    def set_freq(self, freq):
        self.freq = freq

    def set_freq2(self, freq2):
        self.freq2 = freq2

    def get_mse(self, path1, path2):
        fs, y = self.analyzer.read_signal(path1)
        fs2, y2 = self.analyzer.read_signal(path2)

        y = self.analyzer.butter_lowpass_filter(y, self.LOW_CUT_FREQ, fs)
        y2 = self.analyzer.butter_lowpass_filter(y2, self.LOW_CUT_FREQ, fs2)

        y = self.analyzer.divide_signal_into_chunks(y, self.CHUNK_SIZE)
        y2 = self.analyzer.divide_signal_into_chunks(y2, self.CHUNK_SIZE)

        freq = self.analyzer.get_frequencies(y, fs, self.CHUNK_SIZE)
        freq2 = self.analyzer.get_frequencies(y2, fs2, self.CHUNK_SIZE)

        freq, freq2 = self.analyzer.trim_frequencies(freq, freq2)
```

```
        # set freqs to object properties
        self.set_freq(freq)
        self.set_freq2(freq2)

        mse = np.square(np.subtract(freq, freq2)).mean()
        return mse

    # Quadratic function, change small to big, big to small values
    def process_mse(self, mse):
        mse = mse - 50000
        mse = mse / 1000
        mse = np.power(mse, 2)
        mse = round(mse)
        return mse

    def get_frequencies(self):
        return {'freq': self.freq, 'freq2': self.freq2}

    def get_score(self, path1, path2):
        mse = self.get_mse(path1, path2)
        # Apply function, else return big number
        if(50000 > mse >= 0):
            return self.process_mse(mse)
        elif(mse >= 50000):
            return 0
        else:
            return np.power(10, 6)
```

Example usage

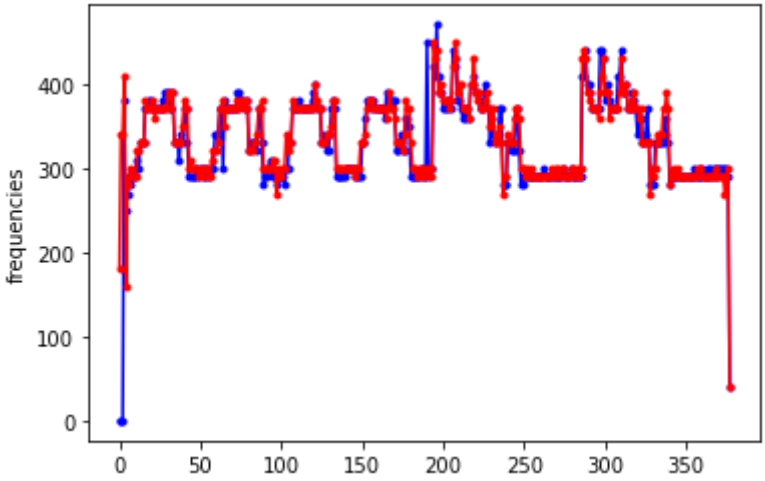
In [ ]:

```
path1 = 'voice_example2_1.wav'
path2 = 'voice_example2_2.wav'

voiceScoreCalculator = VoiceScoreCalculator()
score = voiceScoreCalculator.get_score(path1, path2)
freqs = voiceScoreCalculator.get_frequencies()

freq = freqs['freq']
freq2 = freqs['freq2']
score
```

Plotting result

In [22]:

```
plt.plot(freq, marker=".", color='b')
plt.plot(freq2, marker=".", color='r')
plt.ylabel('frequencies')
plt.show()
```



Plot MSE processing function

- if MSE is small, user gets high score
- if MSE if big, user gets smaller score

In [23]:

```
def processing_fun(x):
    x = x - 50000
    x = x / 1000
    x = np.power(x, 2)
    return x

x = np.arange(0, 50000, 0.1)
y = processing_fun(x)

plt.figure(2)
plt.plot(x, y)
plt.show()
```