<u>**INDIAN INSTITUTE OF TECHNOLOGY, JODHPUR**</u>

SAIDE- AR and VR

PSYCHOPHYSICS AIL7290

ASSIGNMENT 1

Student Details:

Gaurish Garg

M23AIR004

Objective:

To find pitch discrimination and find the difference threshold for a single tone sound using the following Adaptive Staircase Methods

a. 2AFC 1 UP-3 Down Adaptive Staircase method.
b. 3AFC 1 UP-3 Down Adaptive Staircase method.

Parameters Taken:

Standard Parameters:

Standard Frequency (in Hz) = 1000

Duration of sound (in ms) = 250

Other Parameters:

Step Size (in Hz) = 25

Sound Pressure Level = 75

Initial starting frequency for Ascending series = 935

Initial starting frequency for Descending series = 1065

Experiment Procedure:

a. **For 2AFC 1 UP 3 DOWN Adaptive Staircase Method:**

   Before each trial, randomly a series is chosen Ascending or Descending. For each trial, the user is presented with two stimuli, out of which one tone is reference stimulus and the other.is test stimulus. The user is asked which tone is greater. If the user gives an incorrect response, then the test stimulus is sent farther from the reference stimulus by a fixed step size. If the user gives three consecutive correct responses, the test stimulus is brought closer to the reference stimulus by a fixed step size. The transition points where the user changes responses are recorded separately for Ascending and Descending Series. To handle special cases and avoid guessing, the user is also allowed to answer “Equal” if the user feels the stimuli are equal. The code handles “Equal” responses accordingly.

b. **For 3AFC 1 UP 3 DOWN Adaptive Staircase Method:**

   Before each trial, randomly a series is chosen Ascending or Descending. For each trial, the user is presented with three stimuli, out of which two tones are reference stimulus and the other.is test stimulus. The user is asked which tone contains the test stimuli, i.e., which tone is different from the other two. If the user gives an incorrect response, then the test stimulus is sent farther from the reference stimulus by a fixed step size. If the user gives three consecutive correct responses, the test stimulus is brought closer to the reference stimulus by a fixed step size. The transition points where the user changes responses are recorded separately for Ascending and Descending Series. To handle special cases and avoid guessing, the user is also allowed to answer “Equal” if the user feels the stimuli are equal. The code handles “Equal” responses accordingly.

Result Calculation:

The difference threshold is calculated by taking the difference of the average of the Ascending series and average of the Descending Series.

Results:

a. 2AFC 1 UP 3 DOWN Adaptive Staircase Method

Transition Points for Ascending Series:

```
[910, 935, 910, 985, 960, 985, 935, 960, 910, 960, 935, 985]
```

Transition Points for Descending Series:

```
[1040, 1140, 1115, 1140, 1115, 1165, 1090, 1115, 1065, 1090, 1065, 1090, 1065, 1090]
```

Total Number of Trials:

```
98
```

Difference Threshold:

```
77.5
```

Threshold Tracking Curve (Test Stimuli vs number of trials):

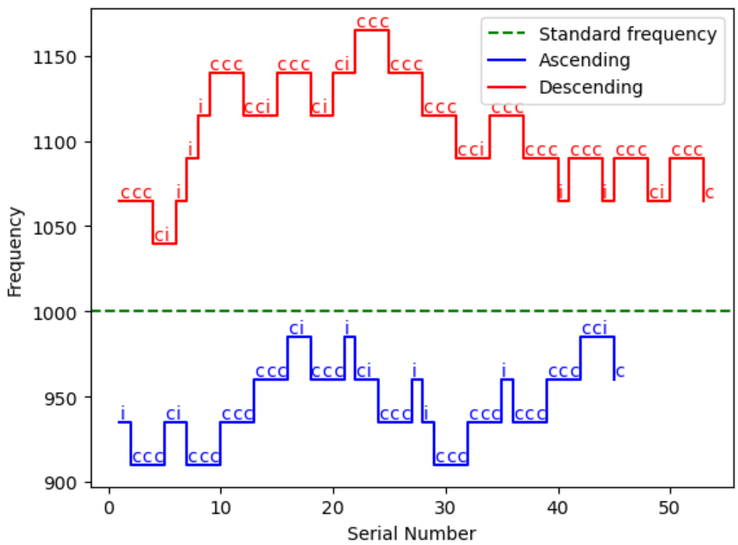

Figure 1

Psychometric Curve: (Proportion Correct vs Change in Stimulus Intensity):

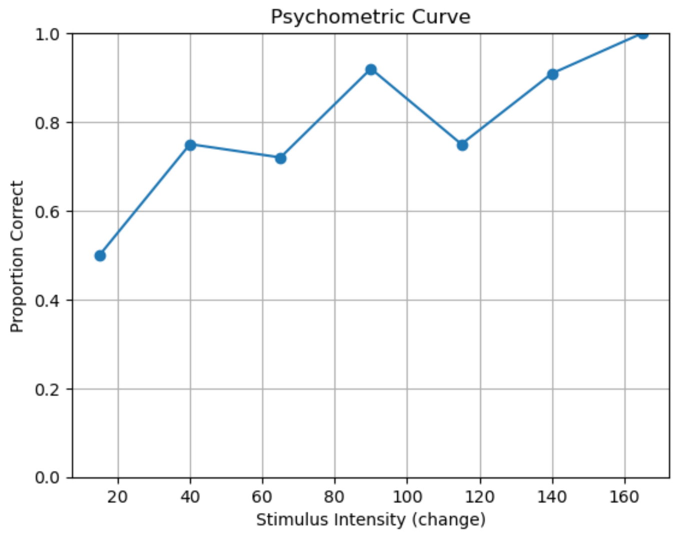

Figure 2

Validation:

The 84.1% point on Y axis on the psychometric curve plotted corresponds to approximately 77.5 point on the X axis.

b. 3AFC 1 UP 3 DOWN Adaptive Staircase Method

Transition Points for Ascending Series:

```
[960, 910, 935, 885, 935, 885, 910, 885, 910, 885, 935, 885, 935, 885]
```

Transition Points for Descending Series:

```
[1165, 1065, 1115, 1065, 1115, 1090, 1115, 1090, 1165, 1090, 1165, 1140]
```

Total Number of Trials:

```
129
```

Difference Threshold:

```
102.5
```

Threshold Tracking Curve (Test Stimuli vs number of trials):

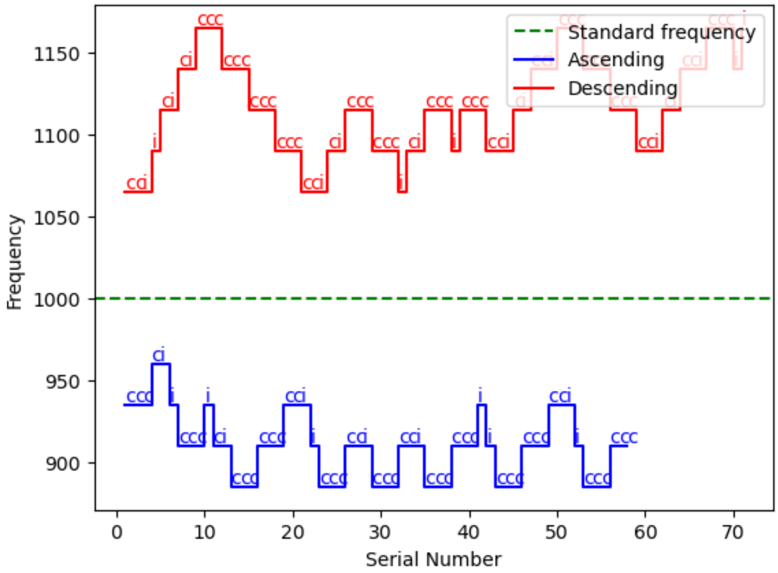

Figure 3

Psychometric Curve: (Proportion Correct vs Change in Stimulus Intensity):

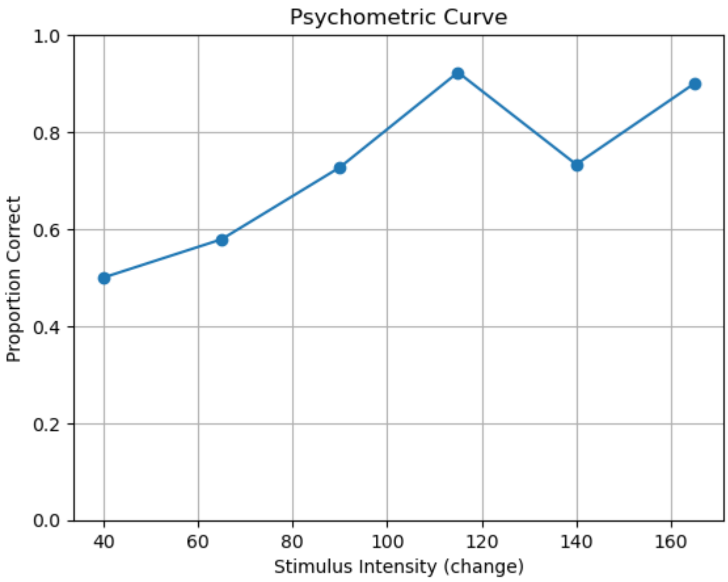

Figure 4

Validation:

The 84.1% point on Y axis on the psychometric curve plotted corresponds to approximately 102.5 point on the X axis.

Code Explanation:

Libraries Required:

Different computation and plotting libraries were required like Numpy and Matplotlib. For playing tones, “Sounddevice” library was used.

```
import numpy as np
import sounddevice as sd
import time
import random
import json
import matplotlib.pyplot as plt
import os
```

Generating Sine Tone:

```
def generate_sinusoidal_tone(frequency, duration, spl, sample_rate):
    num_samples = int((duration / 1000) * sample_rate)
    t = np.linspace(0, duration / 1000, num_samples, False)
    amplitude = 10**((spl - 94) / 20)
    tone = amplitude * np.sin(2 * np.pi * frequency * t)
    return tone
```

- It calculates the number of samples based on the duration, sound pressure level (SPL), and sample rate provided.
- It generates a sinusoidal waveform with the specified frequency and duration using NumPy.
- The amplitude of the tone is adjusted based on the SPL, converting it from dB SPL to linear scale.
- The function returns the generated audio tone as a NumPy array

Testing Audio Devices

```
def test_Audio_devices():
    devices = sd.query_devices()
    print("Available audio devices:")
    for i, device in enumerate(devices):
        if("speaker" in device['name'].lower()):
```

```
            print("Device Number: "+str(i)+" Device Name:
"+device['name'])
    my_sound_device = devices[int(input("Enter the number
corresponding to the audio device you want to use"))]

    try:
        test_tone = generate_sinusoidal_tone(1000, 250, 65,
my_sound_device['default_samplerate'])
        sd.play(test_tone, device=my_sound_device['index'])
        sd.wait()
        print(my_sound_device)
        user_value = int(input("Did you hear?\t 1 for yes 0 for
no"))
        if(user_value == 1):
            return my_sound_device
        else:
            print("Choose another audio device\t")
            return test_Audio_devices()
    except Exception as e:
        print(e)
        print("There was an error testing this device. Choose
another device")
        return test_Audio_devices()
```

The code first checks for Audio devices and prompts the user to select an audio device so that the experiment runs smoothly.

Playing Different Stimuli

a. **2AFC 1UP 3 DOWN Adaptive Staircase Method:**

For each trial, the user is presented with two stimuli, out of which one tone is reference stimulus and the other.is test stimulus. The user is asked which tone is greater. The code also handles responses when the user responds “Equal”. The presentation of stimuli is completely random.

```
def play_2AFC_stimuli(standard_frequency, test_frequency):
    frequencies = [test_frequency, standard_frequency]
```

```
    random.shuffle(frequencies)
    tones = []
    for i,freq in enumerate(frequencies):
        time.sleep(2)
        print("Playing frequency "+str(i))
        test_tone_1 = generate_sinusoidal_tone(freq, duration,
SPL, selected_sound_device['default_samplerate'])
        play_tone(test_tone_1)
        tones.append(test_tone_1)
    greater_response = frequencies.index(max(frequencies))
    if(standard_frequency == test_frequency):
        greater_response = 9
    return greater_response, test_frequency,
tones[frequencies.index(test_frequency)].tolist()
```

b. 3AFC 1UP 3 DOWN Adaptive Staircase Method:

For each trial, the user is presented with three stimuli, out of which two tones are the reference stimuli and the other.is test stimulus. The user is asked which tone is test stimulus, or which tone is different from the other two. The code also handles responses when the user responds "Equal". The presentation of stimuli is completely random

```
def play_3AFC_stimuli(reference_frequency, test_frequency):



    test_tone = []
    frequencies = [reference_frequency, test_frequency,
reference_frequency]
    random.shuffle(frequencies)

    for i, myfreq in enumerate(frequencies):
        time.sleep(2)
        print("Playing frequency "+str(i))
```

```
        test_tone_1 = generate_sinusoidal_tone(myfreq,
duration, SPL, selected_sound_device['default_samplerate'])
        play_tone(test_tone_1)
        if(myfreq == test_frequency):
            test_tone = test_tone_1.tolist()

    time.sleep(2)
    correct_response = frequencies.index(test_frequency)
    if(reference_frequency == test_frequency):
        correct_response = 9
    return correct_response, test_frequency, test_tone
```

Logic for Correct and Incorrect Responses:

If the user gives an incorrect response, then the test stimulus is sent farther from the reference stimulus by a fixed step size. If the user gives three consecutive correct responses, the test stimulus is brought closer to the reference stimulus by a fixed step size. The transition points where the user changes responses are recorded separately for Ascending and Descending Series. To handle special cases and avoid guessing, the user is also allowed to answer “Equal” if the user feels the stimuli are equal. The code handles “Equal” responses accordingly. A code snippet for handling incorrect responses and correct responses in Descending series for 2AFC 1 UP 3 DOWN Adaptive Staircase Method is given below:

Incorrect:

```
def
last_response_is_incorrect_asc_2AFC(last_response_of_selected_series
):
    global step_size
    global SPL
    global duration
    global standard_freq
    global trials_2AFC
    global initially_below_reversal_count_2AFC
    global step_size_record_correct
    global step_size_record_incorrect
    global min_step_size
    global max_step_size
```

```
    test_frequency =
last_response_of_selected_series["test_frequency"]



    if(last_response_of_selected_series["test_frequency"] <
standard_freq):
        test_frequency =
last_response_of_selected_series["test_frequency"] - step_size
    elif(last_response_of_selected_series["test_frequency"] >
standard_freq):
        test_frequency =
last_response_of_selected_series["test_frequency"] + step_size
    else:
        test_frequency =
last_response_of_selected_series["test_frequency"] - step_size



    correct_response, test_frequency, test_tone =
play_2AFC_stimuli(standard_freq, test_frequency)

    trials_2AFC = trials_2AFC + 1
    user_input = get_user_response_2AFC()

    if(user_input==correct_response):


        test_case_1 = {"test_tone": test_tone, "test_frequency":
test_frequency, "user_response":"correct"}
        initially_below_2AFC.append(test_case_1)
        frequency_for_initially_below_2AFC.append(test_frequency)
        map_For_initially_below_2AFC.append({"frequency":
test_frequency, "user_response": "correct"})

    else:
```

```
        test_case_1 = {"test_tone": test_tone, "test_frequency":
test_frequency, "user_response":"incorrect"}
        initially_below_2AFC.append(test_case_1)
        frequency_for_initially_below_2AFC.append(test_frequency)
        map_For_initially_below_2AFC.append({"frequency":
test_frequency, "user_response": "incorrect"})
```

Correct:

```
def
last_response_is_correct_asc_2AFC(last_response_of_selected_series):
    global correct_count_initially_below_2AFC
    global step_size
    global SPL
    global duration
    global standard_freq
    global trials_2AFC
    global initially_below_reversal_count_2AFC
    global step_size_record_correct
    global step_size_record_incorrect
    global min_step_size
    global max_step_size

    if(correct_count_initially_below_2AFC < 3 and
correct_count_initially_below_2AFC >=1):

        test_frequency =
last_response_of_selected_series['test_frequency']
        correct_response, test_frequency, test_tone =
play_2AFC_stimuli(standard_freq, test_frequency)



        user_input = get_user_response_2AFC()

        trials_2AFC = trials_2AFC + 1
```

```
        if (user_input!=correct_response):

            test_case_1 = {"test_tone": test_tone, "test_frequency":
test_frequency, "user_response":"incorrect"}
            frequency_for_initially_below_2AFC.append(test_frequency
)
            initially_below_2AFC.append(test_case_1)
            incorrects_between_consecutive_corrects_asc_2AFC.append(
test_case_1)
            map_For_initially_below_2AFC.append({"frequency":
test_frequency, "user_response": "incorrect"})

            correct_count_initially_below_2AFC = 0


        else:

            correct_count_initially_below_2AFC += 1
            test_case_1 = {"test_tone": test_tone, "test_frequency":
test_frequency, "user_response":"correct"}
            initially_below_2AFC.append(test_case_1)
            frequency_for_initially_below_2AFC.append(test_frequency
)
            map_For_initially_below_2AFC.append({"frequency":
test_frequency, "user_response": "correct"})
        return
    elif(correct_count_initially_below_2AFC == 0):




        test_frequency =
last_response_of_selected_series["test_frequency"]
```

```
        if(last_response_of_selected_series["test_frequency"] <
standard_freq):
            test_frequency =
last_response_of_selected_series["test_frequency"] + step_size
        elif(last_response_of_selected_series["test_frequency"] >
standard_freq):
            test_frequency =
last_response_of_selected_series["test_frequency"] - step_size
        else:
            test_frequency =
last_response_of_selected_series["test_frequency"] + step_size



        correct_response, test_frequency, test_tone =
play_2AFC_stimuli(standard_freq, test_frequency)



        user_input = get_user_response_2AFC()

        trials_2AFC = trials_2AFC + 1

        if (user_input==correct_response):


            test_case_1 = {"test_tone": test_tone, "test_frequency":
test_frequency, "user_response":"correct"}
            initially_below_2AFC.append(test_case_1)
            frequency_for_initially_below_2AFC.append(test_frequency
)
            map_For_initially_below_2AFC.append({"frequency":
test_frequency, "user_response": "correct"})
```

```
        else:

            test_case_1 = {"test_tone": test_tone, "test_frequency":
test_frequency, "user_response":"incorrect"}

            initially_below_2AFC.append(test_case_1)
            frequency_for_initially_below_2AFC.append(test_frequency
)
            map_For_initially_below_2AFC.append({"frequency":
test_frequency, "user_response": "incorrect"})
```

Logic Handling and Explanation:

The code contains various global variables and empty lists to store data related to the 2AFC and 3AFC experiments. Two types of series are used, one starting below a reference frequency (Ascending Series) and the other starting above the reference frequency (Descending Series). Randomly a series is chosen for a trial as shown in the following code snippet for running 2AFC trials. Similar logic for randomisation was followed for 3AFC trials. Each trial involves presenting auditory stimuli with specific frequencies and duration and recording the user's response. Data for each trial, including the test tone, test frequency, and user response, are stored in separately. Correct and incorrect responses are tracked, and counts are used to handle transitions between consecutive correct responses. Changes in the test frequency are made based on user responses as per logic described above.

```
initially_below_reversal_count_2AFC = 0
initially_above_reversal_count_2AFC = 0
def run_2afc():
    global reversals_for_initially_above_2AFC
    global reversals_for_initially_below_2AFC
    global initially_above_reversal_count_2AFC
    global initially_below_reversal_count_2AFC
    choice = np.random.randint(0,2)
    if(choice == 0):
```

```
        initially_above_2afc_trial()
        reversals_for_initially_above_2AFC =
find_transition_points(frequency_for_initially_above_2AFC)
        initially_above_reversal_count_2AFC =
len(reversals_for_initially_above_2AFC)

    else:
        initially_below_2afc_trial()
        reversals_for_initially_below_2AFC =
find_transition_points(frequency_for_initially_below_2AFC)
        initially_below_reversal_count_2AFC =
len(reversals_for_initially_below_2AFC)
def run_2afc_experiment():

    global initially_below_reversal_count_2AFC
    global initially_above_reversal_count_2AFC
    while(initially_below_reversal_count_2AFC< 12 or
initially_above_reversal_count_2AFC < 12):
        run_2afc()
    print("End of Experiment 2AFC")
```

Finding Transition Points:

During every trial, the frequencies recorded so far are used to determine the transition points separately for Ascending and Descending series. The transition points are when the series changes from going farther from reference stimulus to going towards the reference stimulus, or vice versa.

```
def find_transition_points(frequency_array):
    # Initialize variables to track direction and transition points
    direction = None  # 'up', 'down', or None (initial state)
    transition_points = []

    # Iterate through the frequency array to find transitions
    for i in range(1, len(frequency_array)):
        current_frequency = frequency_array[i]
        previous_frequency = frequency_array[i - 1]
```

```
        if current_frequency > previous_frequency:
            new_direction = 'up'
        elif current_frequency < previous_frequency:
            new_direction = 'down'
        else:
            new_direction = direction  # Use the previous direction
for repeated values

        if direction is None:
            direction = new_direction
        elif direction != new_direction:
            # Direction changed, record the transition point
            transition_points.append(previous_frequency)
            direction = new_direction

    # Print the transition points
    return transition_points
```

Plotting the results:

The data recorded was stored in JSON files and then loaded. Matplotlib library was used to plot the results.

Computing Proportion of Correct Responses for each change in stimulus frequency:

```
combined_list = list1+list2
combined_list.sort(key=lambda x: x["frequency"])

total_instances = []
for values in combined_list:
    frequency = values["frequency"]
    response = values["user_response"]
    change = abs(values["frequency"] - standard_freq)

    if(response == "correct"):
```

```
        if(not value_exists(change,total_instances)):

            total_instances.append({"change": change,
"instances_count": 1, "correct_count": 1})

        elif(value_exists(change,total_instances)):
            for myvalue in total_instances:
                if(myvalue["change"] == change):
                    myvalue["correct_count"] =
myvalue["correct_count"]+1
                    myvalue["instances_count"] =
myvalue["instances_count"]+1

    else:
        if(not value_exists(change,total_instances)):
            total_instances.append({"change":change,
"instances_count": 1, "correct_count": 0})
        else:
            for myvalue in total_instances:
                if(myvalue["change"] == change):
                    myvalue["instances_count"] =
myvalue["instances_count"]+1
total_instances.sort(key=lambda x: x["change"])
plt.plot([entry["change"] for entry in total_instances],
[entry["correct_count"]/entry["instances_count"] for entry in
total_instances], marker='o', linestyle='-')
plt.xlabel('Stimulus Intensity (change)')
plt.ylabel('Proportion Correct')
plt.title('Psychometric Curve')
plt.grid(True)
plt.ylim(0, 1)  # Set the y-axis limits to [0, 1]
plt.show()
print(total_instances)
```

The `frequency` and `response` values are extracted from the item.

The change in stimulus intensity is calculated by subtracting the standard frequency from the item's frequency.

If the response is correct, then:

> If the change in stimulus intensity does not already exist in the `total_instances` list, then a new entry is created for it. This entry will store the change in stimulus intensity, the total number of instances, and the number of correct instances.
>
> If the change in stimulus intensity already exists in the `total_instances` list, then the number of correct instances for that entry is incremented.

If the response is incorrect, then:

> If the change in stimulus intensity does not already exist in the `total_instances` list, then a new entry is created for it. This entry will store the change in stimulus intensity, the total number of instances, and the number of correct instances (which will be 0).
>
> If the change in stimulus intensity already exists in the `total_instances` list, then the total number of instances for that entry is incremented.

****END OF REPORT****