Battu-s4120310-ModelExperiment

February 10, 2020

[2]:
```
import numpy as np
import pandas as pd
from model import Model
from dmchunk import Chunk
import math
import random
```

[3]:
```
# The timing functions

def noise(s):
    rand = random.uniform(0.001,0.999)
    return s * math.log((1 - rand)/rand)


def time_to_pulses(time, t_0 = 0.011, a = 1.1, b = 0.015, add_noise = True):

    pulses = 0
    pulse_duration = t_0

    while time >= pulse_duration:
        time = time - pulse_duration
        pulses = pulses + 1
        pulse_duration = a * pulse_duration + add_noise * noise(b * a *␣
 ↪pulse_duration)

    return pulses


def pulses_to_time(pulses, t_0 = 0.011, a = 1.1, b = 0.015, add_noise = True):

    time = 0
    pulse_duration = t_0

    while pulses > 0:
        time = time + pulse_duration
        pulses = pulses - 1
```

```
        pulse_duration = a * pulse_duration + add_noise * noise(b * a *␣
↪pulse_duration)

    return time
```

[4]:
```
shortBiased = 30  #ShortBiased Subjects
longBiased = 30  #LongBiased Subjects
Block1_Trials = 80 #Block 1 ends at a total of 80 Trials
Block2_Trials = 160 #Block 2 ends at a total of 160 Trials, i.e from 80 to 160.
```

[147]:
```
#The defined durations that will be shown to the subjects
durations = [1165,1265,1395,1535,1675]
```

[324]:
```
#The dataframes to collect the model data
modelled_data_short = pd.DataFrame(columns =␣
↪['workerId','count_Trial','dur','block','rt','title'])
modelled_data_long = pd.DataFrame(columns =␣
↪['workerId','count_Trial','dur','block','rt','title'])
```

[325]:
```
#Collecting distribution of durations for trials over all subjects(Extra␣
↪information)
trial_data = [0 for i in range(160)]
```

[326]:
```
#Used to check if its currently a Group A subject or not
is_GroupA = True

count = 0

lock=0
#Iterating over each subject
for subject in range(shortBiased+longBiased):

    #Use lock to prevent this if statement from running again after subjects␣
↪>=30
    if subject >= 30 and lock==0:

        is_GroupA = False
        count = 0
        lock=1

    #Each subject is a model
    m = Model()

    # Block 1
    frequencies = [16,16,16,16,16,16]
    duration_index = [0,1,2,3,4]
    #We run the trials here from 0-79 (80 trials)
    for trial in range(Block1_Trials):
```

```
        # Select a random choice whose frequency > 0
        while True:
            selection = np.random.choice(duration_index)
            if frequencies[selection] <= 0:
                duration_index.remove(selection)
                continue
            else:
                break
        frequencies[selection] -= 1

        #Initial Fixation point
        m.time += 3 + np.random.randint(-250,250)/1000
        #Get the duration to show
        current_duration = durations[selection]

        #Collect trial data for either short duration or long duration(Extra␣
↪information)
        if is_GroupA:
            trial_data[trial] += current_duration
            #pass
        else:
            #trial_data[trial] += current_duration
            pass

        #Convert to pulses and add encounter
        pulses = time_to_pulses(current_duration/1000)
        fact = Chunk(name = "tf" + str(pulses), slots ={"isa":"timing-fact",␣
↪"pulses" : pulses})
        m.time += current_duration/1000
        m.add_encounter(fact)

        #Second fixation point
        m.time += 0.8 + np.random.randint(-100,100)/1000
        #Retrieve the memory and wait for that duration and record
        pattern = Chunk(name = "retrieval", slots = {"isa":"timing-fact"})
        t,latency = m.retrieve_blended_trace(pattern, "pulses")
        m.time += latency
        reproduced_time = pulses_to_time(t)
        m.time += reproduced_time
        reproduced_time = (reproduced_time)*1000


        #Add to short data or long data depending on group, and add a very␣
↪minor delay
        if is_GroupA:
```

```
            modelled_data_short.loc[count] =␣
↪[subject,trial,current_duration,1,reproduced_time,"TimingShort"]
            m.time += 0.015
        else:

            modelled_data_long.loc[count] =␣
↪[subject,trial,current_duration,1,reproduced_time,"TimingLong"]
            m.time += 0.015
        count += 1


    #break time taken between block 1 and block 2
    m.time += 120
    #Block 2
    long_frequencies = [12,12,12,12,32]
    short_frequencies = [32,12,12,12,12]
    duration_index = [0,1,2,3,4]

    #For short
    if is_GroupA:
        #Trial runs from 80-159 (80 trials total)
        for trial in range(Block1_Trials,Block2_Trials):
            #Select duration whose frequency >0
            while True:
                selection = np.random.choice(duration_index)
                if short_frequencies[selection] <= 0:
                    duration_index.remove(selection)
                    continue
                else:
                    break
            short_frequencies[selection] -= 1

            #First fixation point
            m.time += 3 + np.random.randint(-250,250)/1000
            current_duration = durations[selection]

            #Adding data to trial
            trial_data[trial] += current_duration

            #Convert to pulses and store encounter
            pulses = time_to_pulses(current_duration/1000)
            fact = Chunk(name = "tf" + str(pulses), slots ={"isa":
↪"timing-fact", "pulses" : pulses})
            m.time += current_duration/1000
            m.add_encounter(fact)

            #Second fixation point
```

```
            m.time += 0.8 + np.random.randint(-100,100)/1000
            pattern = Chunk(name = "retrieval", slots = {"isa":"timing-fact"})
            t,latency = m.retrieve_blended_trace(pattern, "pulses")
            m.time += latency
            reproduced_time = pulses_to_time(t)
            m.time += reproduced_time
            reproduced_time = (reproduced_time)*1000

            modelled_data_short.loc[count] =␣
↪[subject,trial,current_duration,2,reproduced_time,"TimingShort"]
            m.time += 0.015
            count +=1

    #For long biased
    else:
        #Run trials from 80-159 (80 total trials)
        for trial in range(Block1_Trials,Block2_Trials):
            #Select duration with frequency>0
            while True:

                selection = np.random.choice(duration_index)
                if long_frequencies[selection] <= 0:
                    duration_index.remove(selection)
                    continue
                else:
                    break

            long_frequencies[selection] -= 1

            #First fixation point
            m.time += 3 + np.random.randint(-250,250)/1000
            current_duration = durations[selection]

            #adding to trial data(Extra data)
            #trial_data[trial] += current_duration

            #Convert to pulses and store in memory
            pulses = time_to_pulses(current_duration/1000)
            fact = Chunk(name = "tf" + str(pulses), slots ={"isa":
↪"timing-fact", "pulses" : pulses})
            m.time += current_duration/1000
            m.add_encounter(fact)

            #Second fixation point
            m.time += 0.8 + np.random.randint(-100,100)/1000
            pattern = Chunk(name = "retrieval", slots = {"isa":"timing-fact"})
            t,latency = m.retrieve_blended_trace(pattern, "pulses")
```

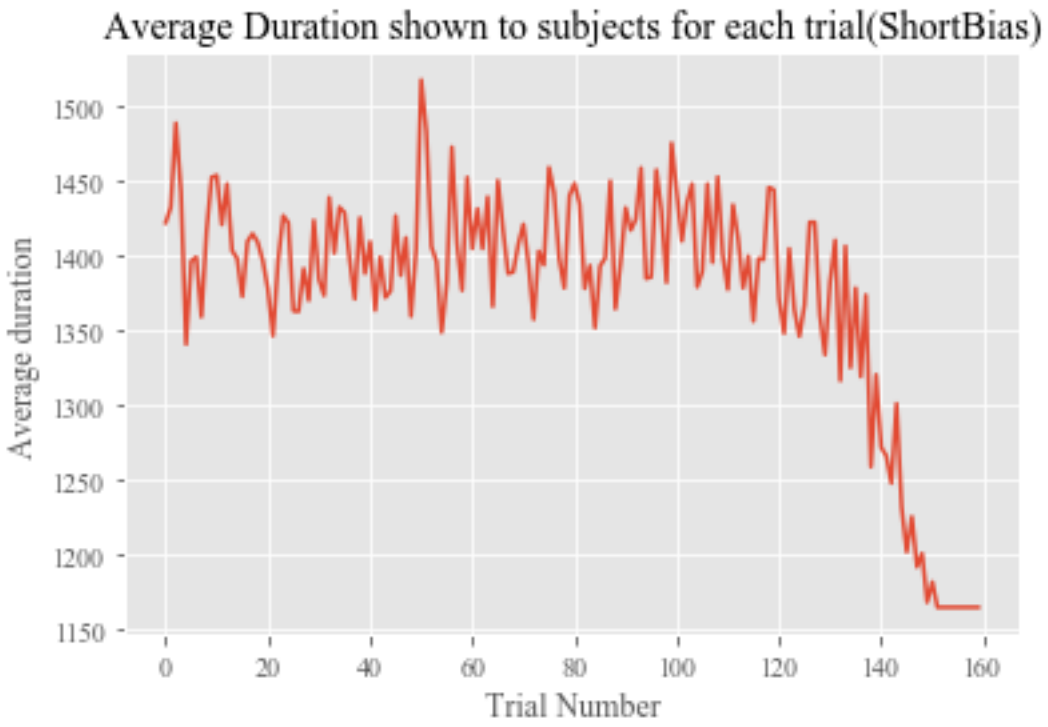
```
            m.time += latency
            reproduced_time = pulses_to_time(t)
            m.time += reproduced_time
            reproduced_time = (reproduced_time)*1000

            modelled_data_long.loc[count] =␣
↪[subject,trial,current_duration,2,reproduced_time,"TimingLong"]
            m.time += 0.015
            count+=1
```

[327]:
```
trial_data = [x/30 for x in trial_data]
```

[329]:
```
plt.rcParams["font.family"] = "Times New Roman"
plt.title("Average Duration shown to subjects for each trial(ShortBias)")
plt.xlabel("Trial Number")
plt.ylabel("Average duration")
plt.plot(range(160),trial_data)
```

[329]:
```
[<matplotlib.lines.Line2D at 0x241fb613f98>]
```



[315]:
```
modelled_data_short.to_csv("data_short.csv")
```

[316]:
```
modelled_data_long.to_csv("data_long.csv")
```

[ ]:
```
# The rest of the analysis done in R
```