Homework2

November 22, 2020

1 Cognitive Modelling: Homework 2

Name: Vishal Sreenivasan
Student No.: S4196392

1.1 Importing files & libraries

[235]:

```
from model import Model
from dmchunk import Chunk
import pandas as pd
import random
import numpy as np
import matplotlib.pyplot as plt
```

1.2 Ready Set Go base function

[228]:

```
def rsg(m, interA, interB):
    #initializing the cues
    cues = ["Ready", "Set", "Go"]
    #adding cues to model dm
    for i in range(len(cues) - 1):
        fact = Chunk(name = "rsg"+cues[i], slots = {"isa": "rsa-fact", "obj1":␣
 ↪cues[i], "obj2": cues[i+1]})
        m.add_encounter(fact)

    #setting goal chunk
    g = Chunk(name = "goal", slots = {"isa": "rsg-goal", "start": cues[0],␣
 ↪"end": cues[-1]})
    m.goal = g
    done = False

    #random delay
    delay_time = random.uniform(0.25,0.85)
    #sample interval
    interval_time = random.uniform(interA, interB)
    while not done:
        if not "current" in g.slots:
            m.time += delay_time
            g.slots["current"] = g.slots["start"]
        elif g.slots["current"] != g.slots["end"]:
            request = Chunk(name = "request", slots = {"isa":"rsa-fact", "obj1":
 ↪ g.slots["current"]})
            chunk, latency = m.retrieve(request)
            g.slots["current"] = chunk.slots["obj2"]
            m.time += latency
            m.time += interval_time
        else:
            done = True
```

1.3 Function to loop for n participants

[229]:

```
def ready_set_go(n, n_training = 500, n_testing = 1000):
    #Setup dataframe
    df = pd.DataFrame(columns = ["Subject", "Cond", "Trial", "T_s", "T_p",␣
 ↪"Main"])

    #variable for counting trials
    trial = 0

    #Three conditions: Long, Intermediate & Short
    conditions = ["One", "Two", "Three"]
    #Sample distribution range
    interval_cond = {"One": [0.494, 0.847],
                     "Two": [0.671, 1.023],
                     "Three": [0.847, 1.20]}

    for cond in conditions:
        for n_i in range(n):
            m = Model() # create a new model for each subject

            for train in range(n_training):
                start_time = m.time
                rsg(m, *interval_cond[cond])

                #converting secs to ms
                f_time = round((m.time - start_time) * 1000, 2)

                data = {"Subject": n_i,
                        "Cond": cond,
                        "Trial": trial,
                        "T_s": f_time,
                        "T_p": f_time,
                        "Main": "Train"}
                df = df.append(data, ignore_index=True)
                trial += 1

            for test in range(n_testing):
                start_time = m.time
                rsg(m, *interval_cond[cond])

                data = {"Subject": n_i,
                        "Cond": cond,
                        "Trial": trial,
                        "T_s": f_time,
                        "T_p": f_time,
                        "Main": "Test"}
                df = df.append(data, ignore_index=True)

                trial += 1

    return df
```

1.4 Run

Model is ran for 5 participants and the data is stored for future analysis

[232]:

```
df = ready_set_go(5)
df.to_csv("hw2.csv", index = False)
```

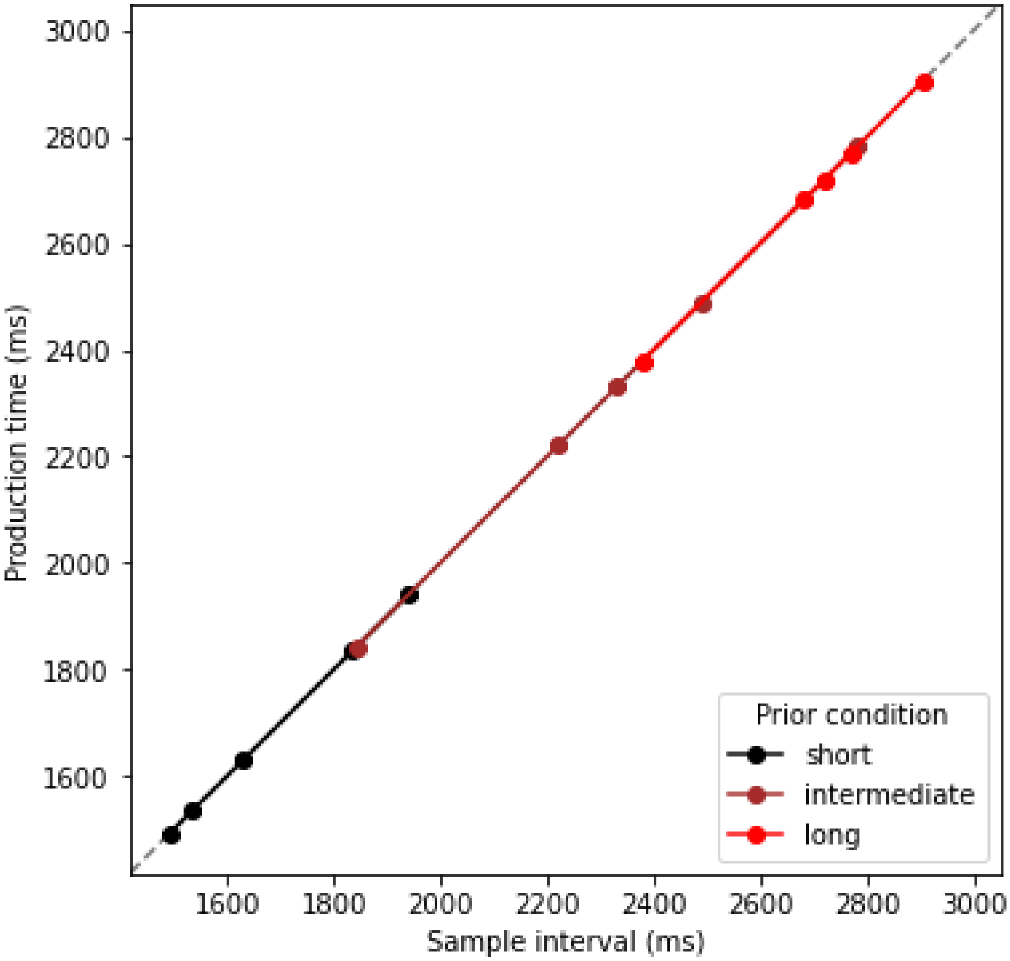
1.5 Plot function (Provided by faculty)

[236]:

```
# Remove training trials
dat = df[df['Main'] == "Test"]

# Calculate mean Tp by condition
mean_tp = dat.groupby(['Cond', 'T_s'])['T_p'].mean().reset_index()

yrange = np.multiply((min(mean_tp['T_s']), max(mean_tp['T_s'])), [0.95, 1.05])

cond1 = mean_tp.loc[mean_tp['Cond'] == "One"]
cond2 = mean_tp.loc[mean_tp['Cond'] == "Two"]
cond3 = mean_tp.loc[mean_tp['Cond'] == "Three"]

# Add jitter noise
jitter = dat.copy()
jitter['T_s'] = jitter['T_s'] + np.random.uniform(-5, 5, len(dat))
cond1_jitter = jitter.loc[jitter['Cond'] == "One"]
cond2_jitter = jitter.loc[jitter['Cond'] == "Two"]
cond3_jitter = jitter.loc[jitter['Cond'] == "Three"]
```

```
# Make plot
f, ax = plt.subplots(figsize = (6,6))

ax.set(xlim = yrange, ylim = yrange)
f.gca().set_aspect('equal', adjustable = 'box')

ax.set_xlabel('Sample interval (ms)')
ax.set_ylabel('Production time (ms)')

ax.plot(yrange, yrange, linestyle = '--', color ='gray')

ax.scatter(cond1_jitter['T_s'], cond1_jitter['T_p'], marker = '.', color =␣
 ↪'black', alpha = 0.025, label = None)
ax.scatter(cond2_jitter['T_s'], cond2_jitter['T_p'], marker = '.', color =␣
 ↪'brown', alpha = 0.025, label = None)
ax.scatter(cond3_jitter['T_s'], cond3_jitter['T_p'], marker = '.', color =␣
 ↪'red', alpha = 0.025, label = None)

ax.plot(cond1['T_s'], cond1['T_p'], color = 'black', marker = 'o', label =␣
 ↪"short")
ax.plot(cond2['T_s'], cond2['T_p'], color = 'brown', marker = 'o', label =␣
 ↪"intermediate")
ax.plot(cond3['T_s'], cond3['T_p'], color = 'red', marker = 'o', label = "long")

ax.legend(title = 'Prior condition', loc = 4)
```

```
[236]: <matplotlib.legend.Legend at 0x1592a0e0bb0>
```


3000
2800
2600
2400
2200
2000
1800
1600
Production time (ms)
1600
1800
2000
2200
2400
2600
2800
3000
Sample interval (ms)
Prior condition
short
intermediate
long