中間課題(課題 1,2)

メソッドエンジニアリング

中央大学理工学部ビジネスデータサイエンス学科

23D7104001I 髙木悠人

1. 課題 1[自動改札機設置問題]

(1) 機能面における要件定義

本課題では、授業内で行った条件下での実施となるが、先に東京都文京区の後楽園駅における改札数の変化による待ち時間のシミュレーションを行い、その上で課題となる条件で実施することとした。そのため、先に本駅の利用者数や運行頻度から本改札に必要とされている機能要件の定義を行う。東京メトロによる各駅の乗降人員ランキングを見ると、2023 年の本駅では 1 日平均 99051 人が利用している。よって、朝 5 時から翌 1 時までの 20 時間稼働すると仮定すると 1 時間あたり平均 5000 人となる。しかし、電車の運行本数は時間帯によって変化するため、時間帯における電車の本数を Python によるウェブスクレイピングで取得しグラフ化した結果以下の通りとなった。ただし、実行したコードは付録の項に掲載する。

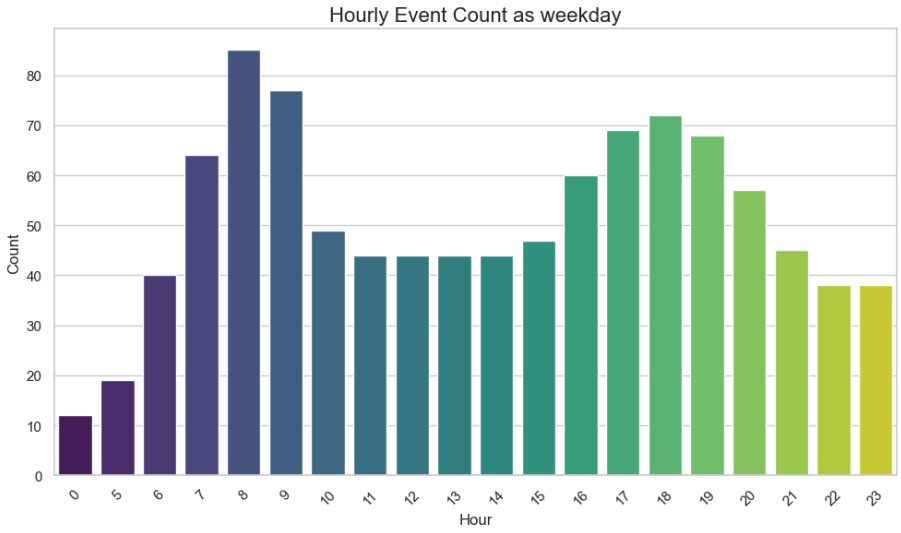

図 1. 平日の時間帯による電車運行本数の推移

南北線と丸の内線それぞれの上下線のデータを参照した。

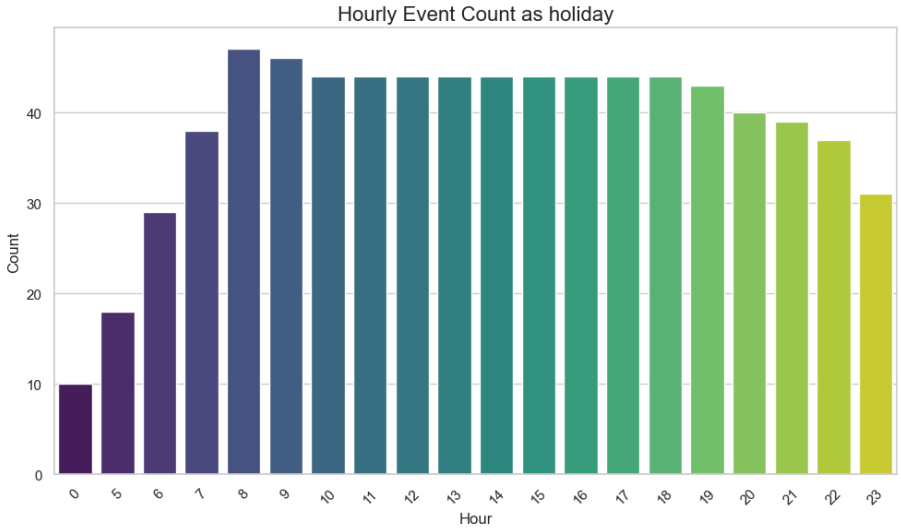

図 2. 休日の時間帯による電車運行本数の推移

上記のグラフを見ると、平日の朝 8 時台が 85 本で最大となり、時間帯による本数の増減が明確になっている。さらに、電車の混雑度も変わってくると推測できる。しかし、オープンデータに電車乗客数がなかったため、混雑度が運行本数に依存すると仮定し当該時間帯の運行本数をN_{train_A}、その日の運行本数をN_{train_B}以下の式で混雑度を定式化した。

$$Con_{time} = \frac{N_{train_A} * 20}{N_{train_B}} * 100 \ [\%]$$

また、平日と休日での混雑度は異なると予測できるため、週の運行本数をN_{sum}とすると以下のように各日における混雑度を定義した。

$$Con_{day} = \frac{N_{train_B} * 7}{N_{sum}} * 100 \ [\%]$$

上記 2 つの式を用いて、各時間帯の利用客数をその混雑率の積に従うと仮定し以下のように仮定した。

$$N_{use} = Con_{day} * Con_{time} * \frac{1}{100} * \frac{99051}{20}$$

よって、最も利用客数が多いとされる平日 8 時台の利用客数を 9607 人と定義した。そしてシミュレーションタイムは、時間帯の区分で利用した 1 時間とすることとした。

(2) シミュレーション

前項の要件定義を用いて、R プログラムでシミュレーションを実施する。ただし、9607 人が 3600 秒間に通過するため、passenger の発生頻度は期待値を

$\frac{3600}{9607} = 0.375[s]$とする指数分布に従うこととする。まず、後楽園駅に現在利用されており、駅内から出る方向の改札数である 7 で実施した結果以下のようになった。

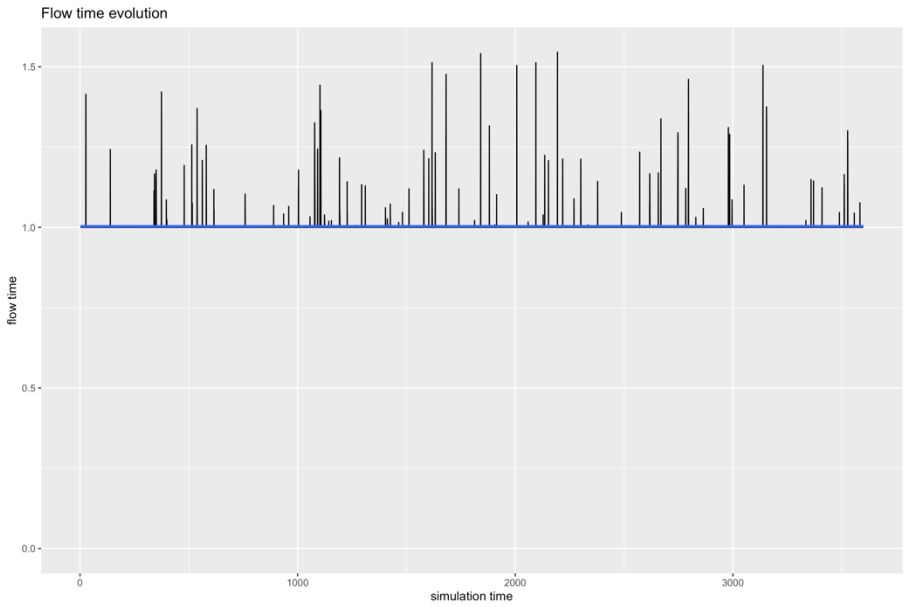

図 3. 改札数を 7 と設定した場合のシミュレーション結果

上記を見ると、待ち時間(改札通過にかかる時間)が約 1.5 秒以内で収まっており、待ち時間の推移を示す青線は 1.0 で直線となっているため、待ち時間がほぼない改札であると言える。次に、この改札数が最適か確かめるため前の改札数= 6 も実施する。実施した結果、図 4 のようになった。

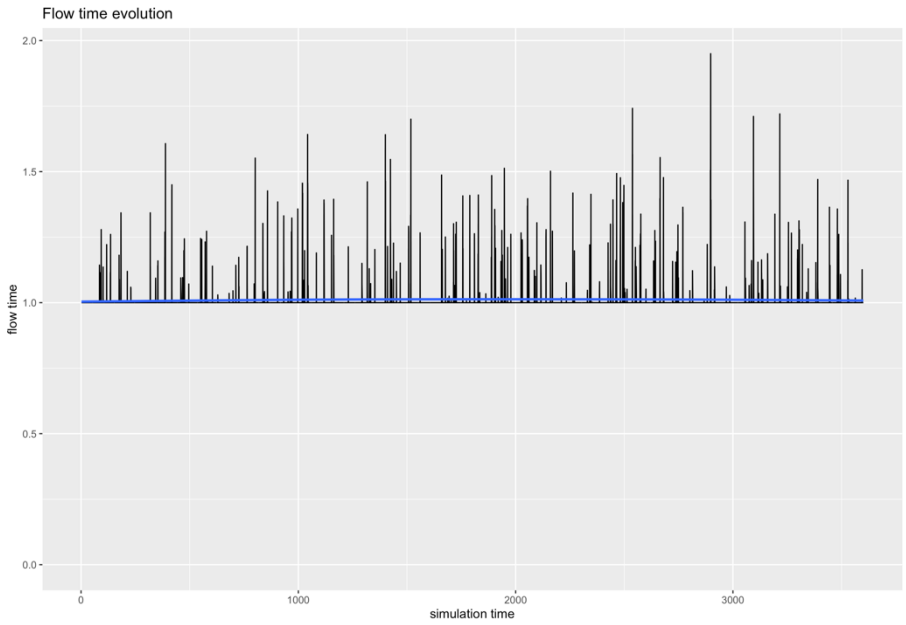

図 4. 改札数を 6 と設定した場合のシミュレーション結果

上記を見ると、青線が y=1.0 と離れている点と待ち時間が 2.0 に近くなっていることから、最適とは言えず改札数を増やすべきであると言える。

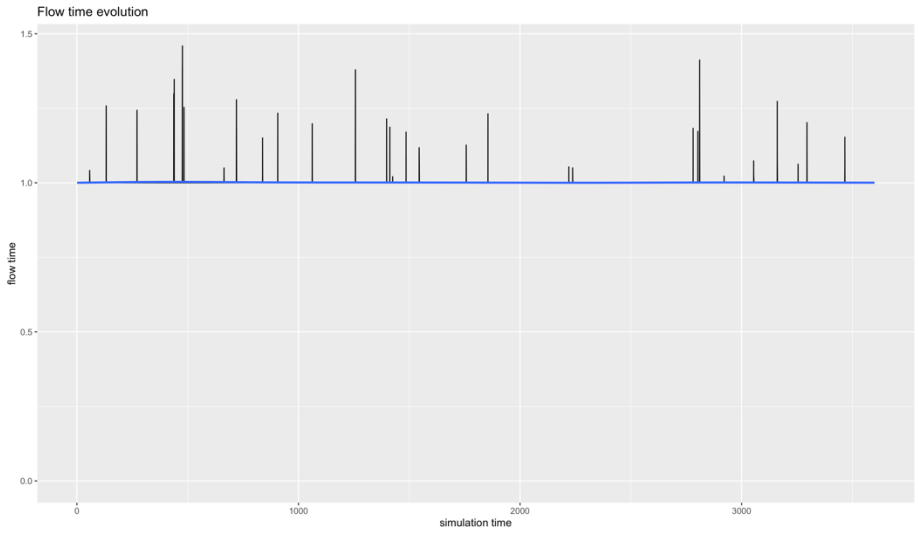

図 4. 改札数を 8 と設定した場合のシミュレーション結果

上記を見ると、最大待ち時間が 1.5 秒ほどとなっていること、改札数を増やしたのにも関わらず待ち時間減少の期待できる効果が薄い点から後楽園の改札数の最適値は 7 であると言える。ただし、改札に入る場合と出る場合があり、そ

れに必要となる時間は 1.0 秒よりも長くなる点から当該駅のように入る状況のみ使用できる改札を 2 つ設置するなどの施策が適していると言える。

(3) 授業内の条件下での最適数について

上記と同様に改札数を変化させていく。まず、期待値 0.1 の指数分布となっており、先の条件よりも小さくなっているため、最適数を 11 から確かめることとする。

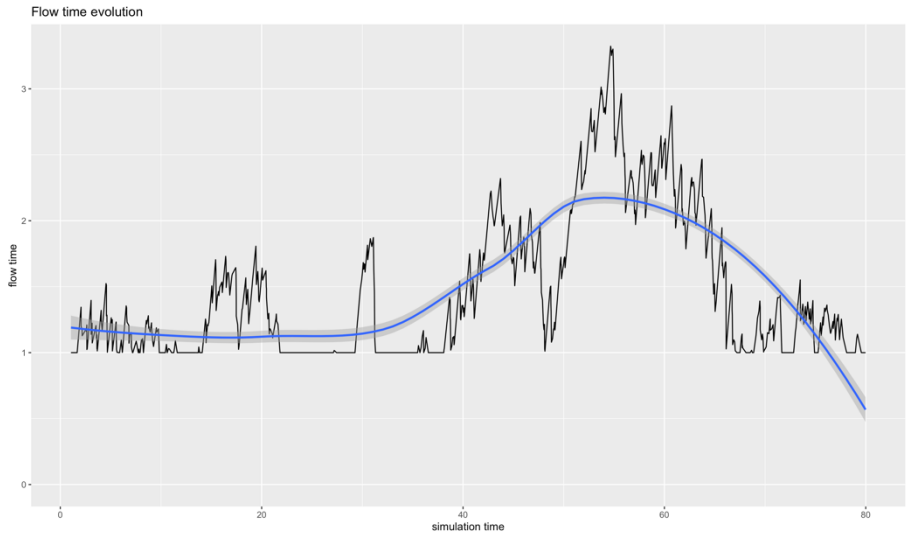

図 4. 改札数を 11 と設定した場合のシミュレーション結果

上記を見ると、改札数が 11 の場合だと待ち時間が 3 秒を超える場合が生じ、長期的に見ると、安定しているとは言えないと推測できるため、改札数を増やすべきであると考える。

よって、改札数を 12 として観察する。結果を図 5 に示す。

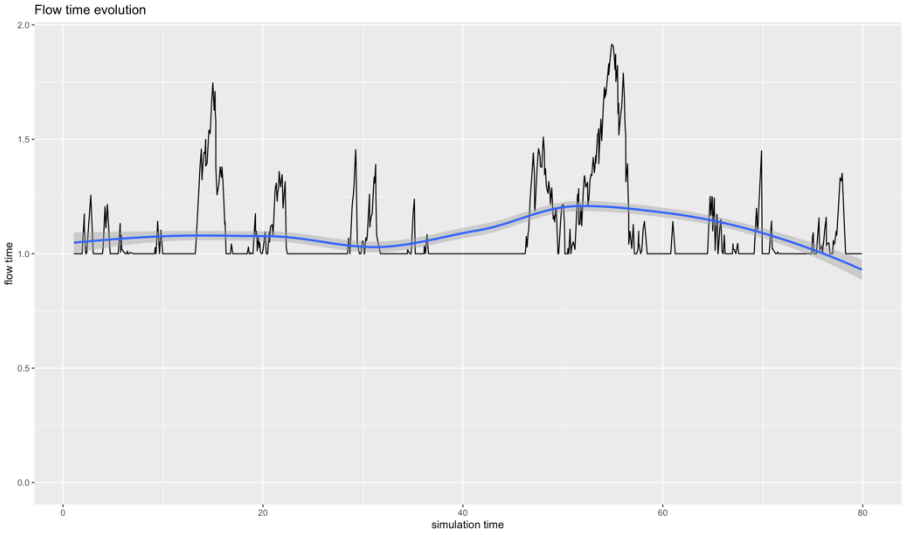

図 5. 改札数を 12 と設定した場合のシミュレーション結果

上記の図を見ると、待ちの発生頻度が顕著に減少していると言える。さらに最大待ち時間が 2.0 秒ほどとなり、良い改札数であると評価できる。しかし、山が複数回生じており、列を成すような待機時間が生じている点、改札数 13 の場合でも顕著な変化が見られる可能性がある点より、確認することとする。図 6 に改札数 13 の結果を示す。

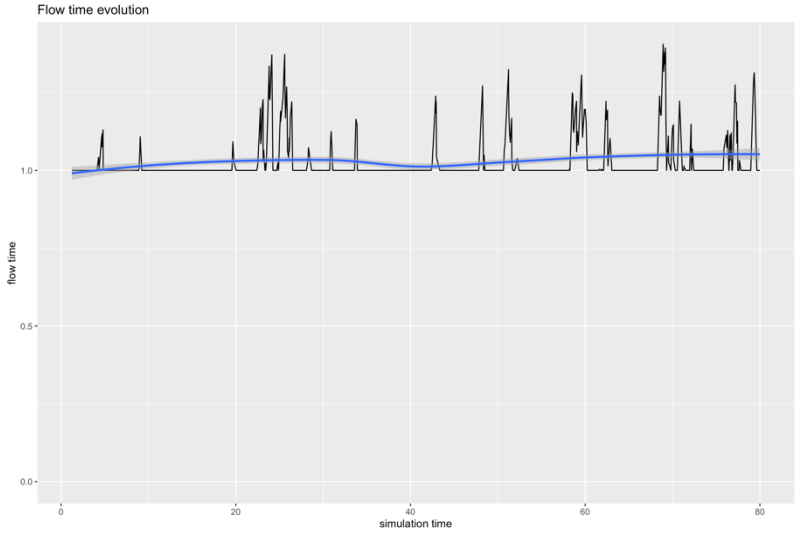

図 6. 改札数を 13 と設定した場合のシミュレーション結果

上記の図を見ると、黒線が山のように盛り上がっている部分がなく、列を成すような待機時間がなくなっていると言える。よって、最適数である可能性が極

めて高いと言える。確認のため、改札数 14 でも実施することとする。

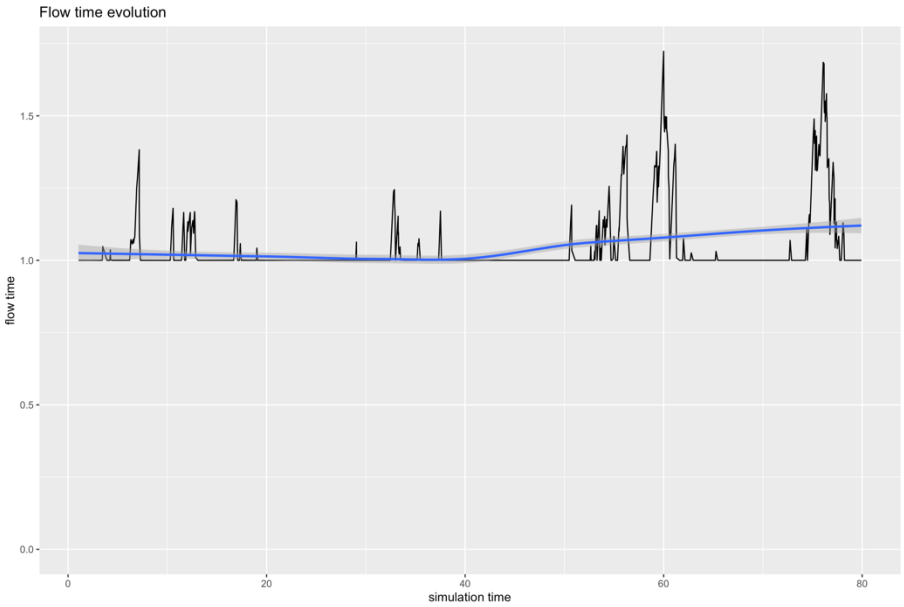

図 7. 改札数を 14 と設定した場合のシミュレーション結果

上記は改札数を 14 と設定したシミュレーション結果である。上記を見ると、山の発生が 13 よりも多くなっている点、近似する青線が y=1 から離れてしまっている点が挙げられる。本事象は改札数を増やしたという条件に対して矛盾しており偶発的な事象であると言えるものの、改札数変化によって期待できる待ち行列解消効果が薄いと考察できるため、最適改札数は、13 であると考える。

2. 課題 2[ラインバランシング]

今回の課題では、まずライン編成効率が 90%を超えるような作業分担を考える。ライン編成効率とは、以下の式で求めることができる。

$$\eta = \frac{\sum_{i=1}^{N} T_i}{N \times T_{max}}$$

よって、編成効率を最適にするには、ネックタイムT_{max}は総作業時間を作業人数で割った値に近づく必要がある。つまり、今回の課題では総時間数 33 分で作業人数が 4 人であるため、8.25 分となる必要がある。したがって、本作業分担では 8 分または、9 分を基準とすることが最適と言える。これを元に、分担した結果以下のように分担できた。

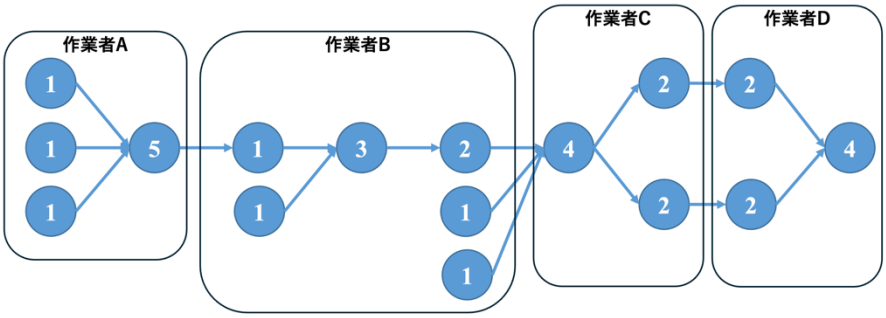

図 8. 最適と言える作業分担

上記を元にすると、以下のような図が作成できる。

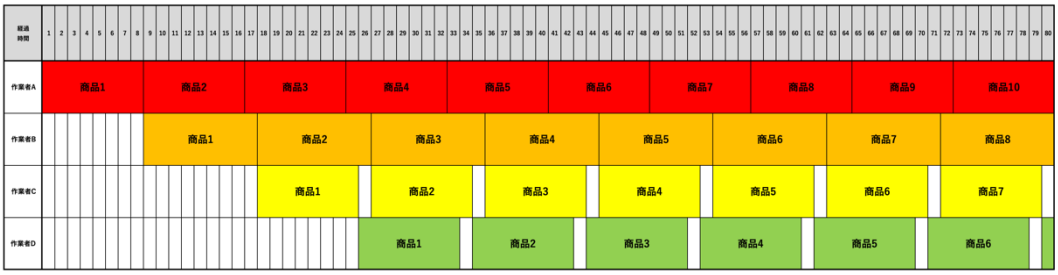

図 9. 作業時間経過図

これを見ると、商品が製造されてから次の商品が製造されるまでの時間は 9 分だとわかるため、サイクルタイムは 9 分であると言える。そこで、作業時間経過図を見ると、作業者 A と作業者 B は休憩なく作業を続けているのにも関わらず、C と D は常に 1 分の休憩を入れているため、負担率に差がある。そして作業者 B の部分で時間を要していることからボトルネックとなっている。つまり、この点で不注意等による事故の発生や過度な疲労が問題視される。よって、本作業工程において作業者を増やすか、作業を再度見直し、短くなるようにすることが改善策として挙げられる。

次に 8 人まで増やす場合について考える。8 人の場合、ライン編成効率を上げるには、各工程の作業時間を 4.1 分に近づけることが有用である。よって、工程を再度分担した結果以下の図を得ることができた。

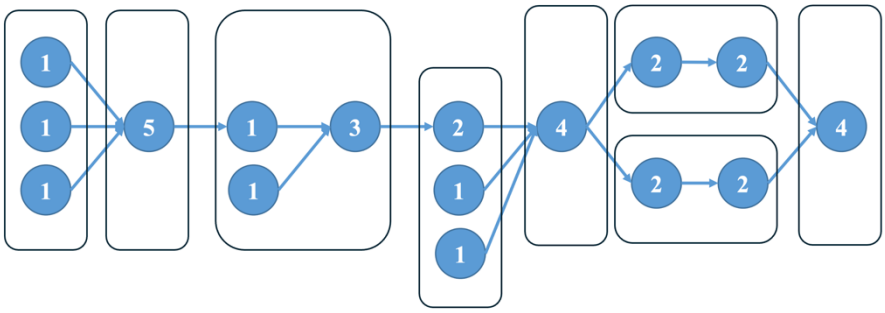

図 10. 最適と言える作業分担

上記の図を元にネックタイムは、各作業者における作業時間の最大値であるため、第 2,3 工程の 5 分となる。そして、サイクルタイムは、ネックタイムに依存するため、5 分と言える。

3. 参考文献

(1) 東京メトロ、「各駅の乗降人員ランキング」、(https://www.tokyometro.jp/corporate/enterprise/passenger_rail/transportation/passengers/index.html)、2024/11/29 参照。

4. 付録

(1) 電車の運行頻度を求める Python Web Scraping コード

```python
import pandas as pd
import requests
from bs4 import BeautifulSoup
import sqlite3
from tqdm.notebook import tqdm
from urllib.request import urlopen, Request
import re
import time
import traceback
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
options = Options()
# その他のクラッシュ対策#-
#options.add_argument("--headless")
# Other crash prevention measures#+
options.add_argument("--no-sandbox")
options.add_argument("--disable-dev-shm-usage")
driver_path = ChromeDriverManager().install()
print("Configuration completed")#+
with webdriver.Chrome(service=Service(driver_path), options=options) as driver:
    driver.implicitly_wait(10)
    M__ = ["M22","N11"]
    direction=[1,0]
    schedule__ = ["weekday","holiday"]
    for sch in schedule__:
        schL=[]
        for M_ in M__:
            for d in direction:
                url = f"https://transfer.tokyometro.jp/website/timetable?numbering={M_}&direction={d}&schedule={sch}"
                time_list_got = []
                try:
                    driver.get(url)
                    time_list = driver.find_elements(By.CLASS_NAME, "departure-time")
                    print(f"時間リスト{len(time_list)}")
                    for time in time_list:
                        time= time.text
                        time_id = re.findall(r"\d{2}:\d{2}",time)
                        time_list_got.append(time_id[0])
                        schL.append(time_id[0])
                except:
                    print(f"stopped at {url}")#-
                    print(traceback.format_exc())
        # 与えられた時刻リスト
        time_list = schL
        # 時刻をDateTimeオブジェクトに変換
        times = pd.to_datetime(time_list, format='%H:%M')
        # 1時間ごとにグループ化するために時間だけを抽出
        hours = times.floor('H')
        # 各時間帯のカウントを計算
        hourly_counts = hours.value_counts().sort_index()
        # データフレームを作成
        df = pd.DataFrame({'Time': hourly_counts.index, 'Count': hourly_counts.values})
        df['Time'] =df['Time'].dt.hour.astype(str)
        display(df)
        # グラフのスタイルを設定
        sns.set_style("whitegrid")
        # Seabornを使ってバープロットを作成
        plt.figure(figsize=(10, 6))
        sns.barplot(x='Time', y='Count', data=df, palette='viridis')
        # グラフのタイトルとラベルを設定
        plt.title(f'Hourly Event Count as {sch}', fontsize=16)
        plt.xlabel('Hour', fontsize=12)
        plt.ylabel('Count', fontsize=12)
        # x軸のラベルを角度をつけて表示
        plt.xticks(rotation=45)
        # グラフを表示
        plt.tight_layout()
        plt.show()
```

(2) シミュレーションコード

```
#install.packages("simmer")
#install.packages("simmer.plot")
#ライブラリのロード
library(simmer)
library(simmer.plot)
get_simulation <- function(gate_num,run_time){

  #simmer のインスタンスを作成する
  env = simmer("station")
  env
  passenger = trajectory("passenger path") %>%
    ## 駅の改札機を定義（通過時間 1 秒）
    seize("gate", 1) %>%
    timeout(1) %>%
    release("gate", 1)
  env %>%
    add_resource("gate", gate_num) %>%
    add_generator("passenger", passenger, function() rexp(1,2.661))
  env %>%
    reset() %>%
    run(run_time) %>%
    now()
  #何人の乗客が通過したか
  env %>% get_n_generated("passenger")
  #それぞれの利用者が何秒目に通過したか
  env %>% get_mon_arrivals()
  #各乗客の待ち時間を確認
  env %>% get_mon_arrivals() %>% transform(waiting_time = end_time -
start_time - activity_time)

  arrivals = get_mon_arrivals(env)
  plot(arrivals, metric = "flow_time")
}
get_simulation(7,3600)
```