

高校情報I
「シミュレーション」
～数理モデル～
Python & Google Colaboratoryによるシミュレーション

高知工科大学 情報学群・データ&イノベーション学群

本日の予定

- ▶ 1時間目：9:00-9:50
 - ▶ モデル化・シミュレーションの説明
 - ▶ 動的モデル（確定的）のシミュレーション
- ▶ 2時間目：10:00-10:50
 - ▶ 1時間目の続き
 - ▶ 動的モデル（確率的）のシミュレーション
- ▶ 3時間目：11:00-11:50
 - ▶ シミュレーションを繰り返す
 - ▶ 様々なパラメータを変えてみる
 - ▶ まとめとふりかえり

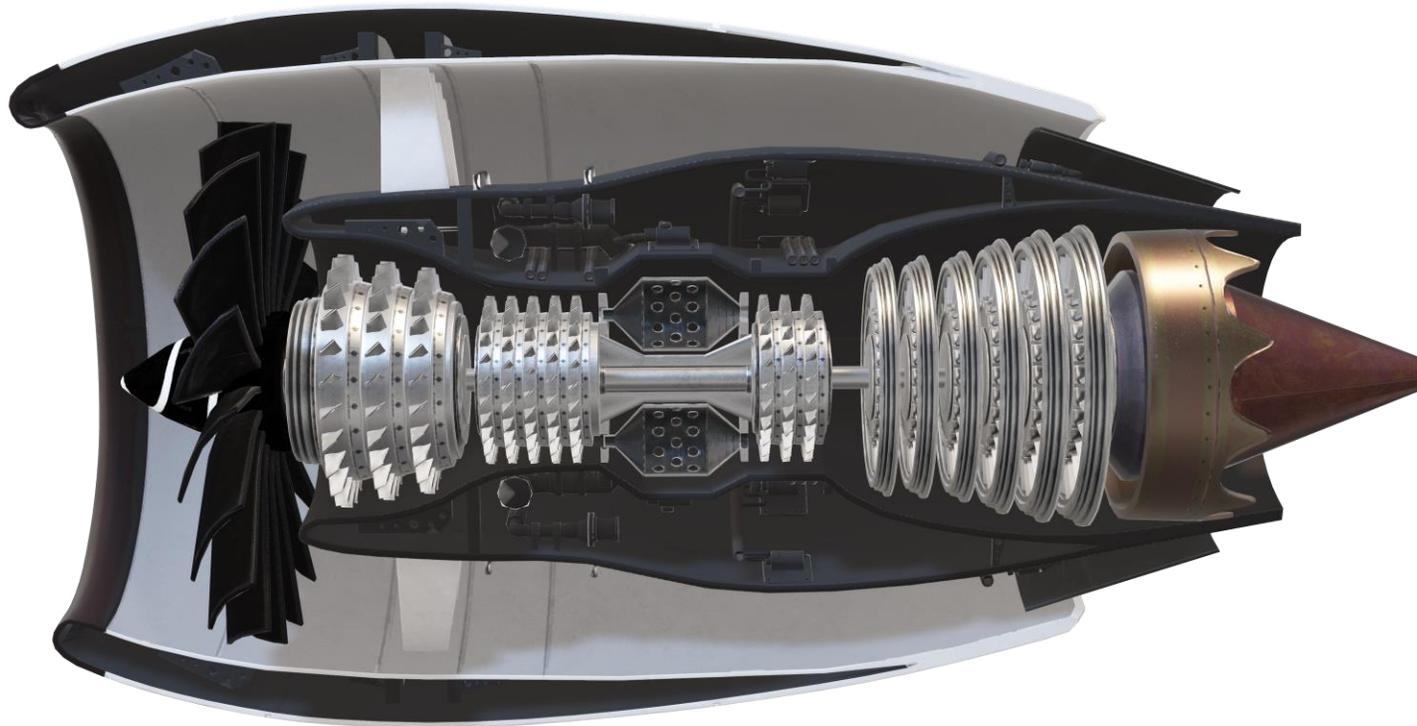
シミュレーションとは

モデル

- ▶ 現実世界を「モデル化」→ 模型
- ▶ モデルの種類
 - ▶ 静的モデル：時間変化しない現象
 - ▶ 動的モデル：時間経過に伴い変化する現象
 - ▶ 確定的モデル：規則的に変化する
 - ▶ 確率的モデル：不規則な変化を含む
- ▶ モデル化の方法
 - ▶ 物理モデル：実際に作る（実物大，拡大，縮小）
 - ▶ 木で飛行機等を作るモックアップ，地球儀，分子のモデル
 - ▶ 図的モデル：図で人・物・情報の流れ，状態の変化を表現
 - ▶ **数理モデル：数式や論理式で対象の状態を記述する**

静的モデルの例

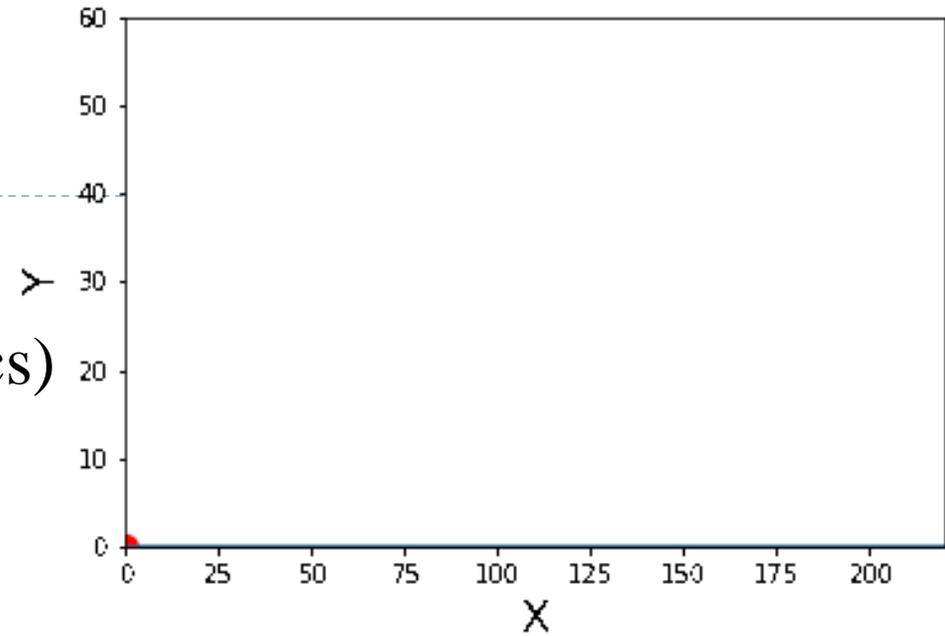
- ▶ ジェットエンジンの3次元モデル
 - ▶ 時間とともに変化しない



ジェットエンジンの3Dモデル
(PowerPoint 3Dモデルより)

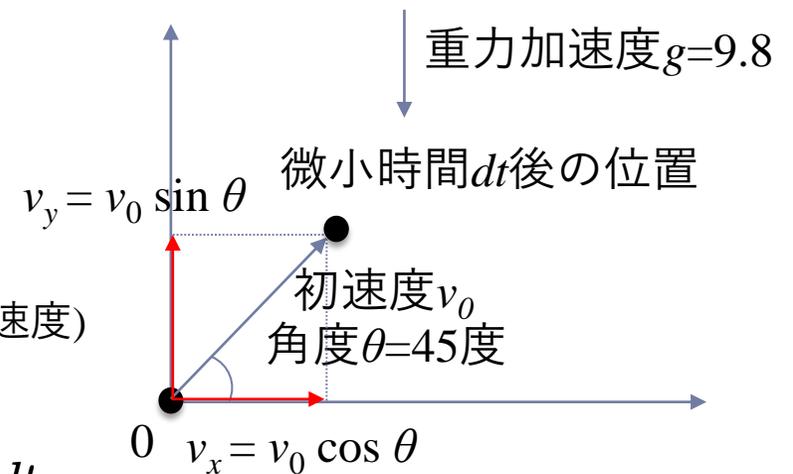
動的モデル(確定的)の例

- ▶ 確定的 (決定的) な例
- ▶ 運動方程式で記述される運動(動き = dynamics)
 - ▶ 放物運動 (ボールの投げ上げ運動)の t 秒後のボールの位置



初速度	$v_0 = 45[\text{m/s}]$
投げ上げ角度	$\theta = 45\text{度}$
重力加速度	$g = 9.8[\text{m/s}^2]$
x方向速度	$v_x(0) = v_0 \cos \theta$
y方向速度	$v_y(0) = v_0 \sin \theta$

微小時間 dt 後のx方向速度	$v_x(t + dt) = v_x(t)$	←速度一定(等速)
微小時間 dt 後のy方向速度	$v_y(t + dt) = v_y(t) - g \times dt$	←速度減少(等加速度)
微小時間 dt 後のx座標位置	$x(t + dt) = x(t) + v_x(t) \times dt$	←右へ進む分
微小時間 dt 後のy座標位置	$y(t + dt) = y(t) + \frac{1}{2} (v_y(t) + v_y(t + dt)) \times dt$	↑上(下)へ進む分



動的モデル(確率的)の例

- ▶ ランダムウォーク
 - ▶ 1秒後に50%の確率で, +1(上)か -1(下)へ移動
 - ▶ t 秒後の位置 S_t は?

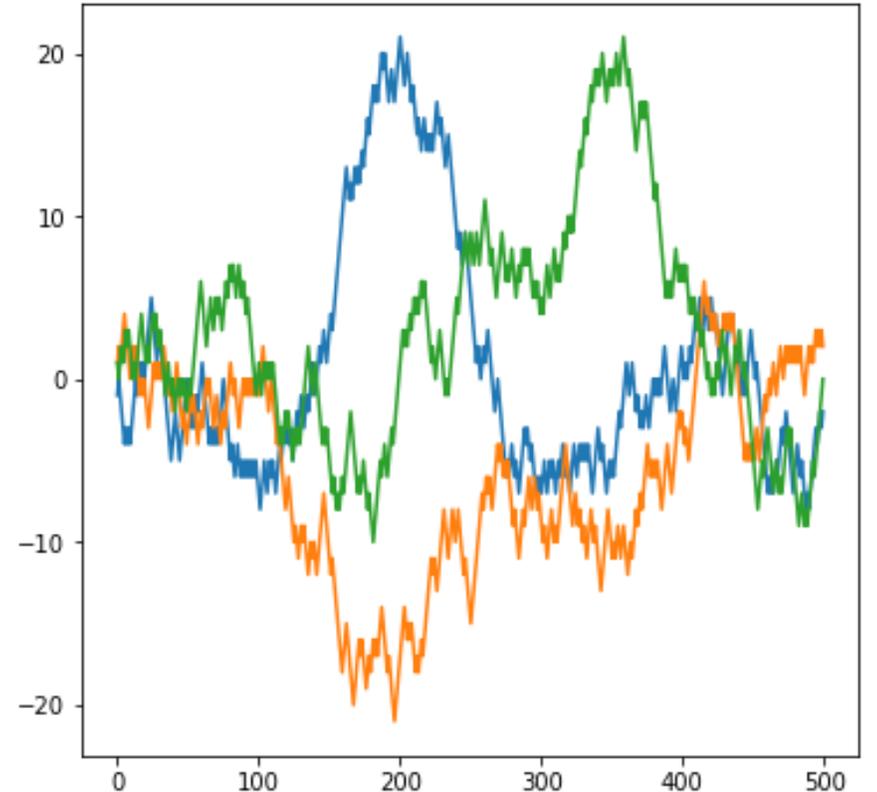
$$S_t = X_1 + X_2 + \cdots + X_t$$

- ▶ ただし, X_i は時刻 i の移動分 +1 or -1

$$P(X_i = 1) = \frac{1}{2}$$

$$P(X_i = -1) = \frac{1}{2}$$

- ▶ いつも決まった値にはならない



シミュレーション

- ▶ シミュレーション：「モデル化」したものを、**模擬的に実行**
- ▶ **実際に実行・観察できないこと**をシミュレーションでできる
 - ▶ 現実では行えないこと，行うにはコストがかかり過ぎることを行う
 - ▶ 例：
 - 交通事故時の車や内部の人
 - 津波到来時の津波の動き
 - 月ができたのは，地球に別の星が衝突したから？
 - ▶ 地球の重力が1/6だったら人や物の動き方はどうなるか
 - ▶ ブラックホールに落ちたらどうなるか
 - ▶ 何度も繰り返せる

語源：

simulation: 模擬実験

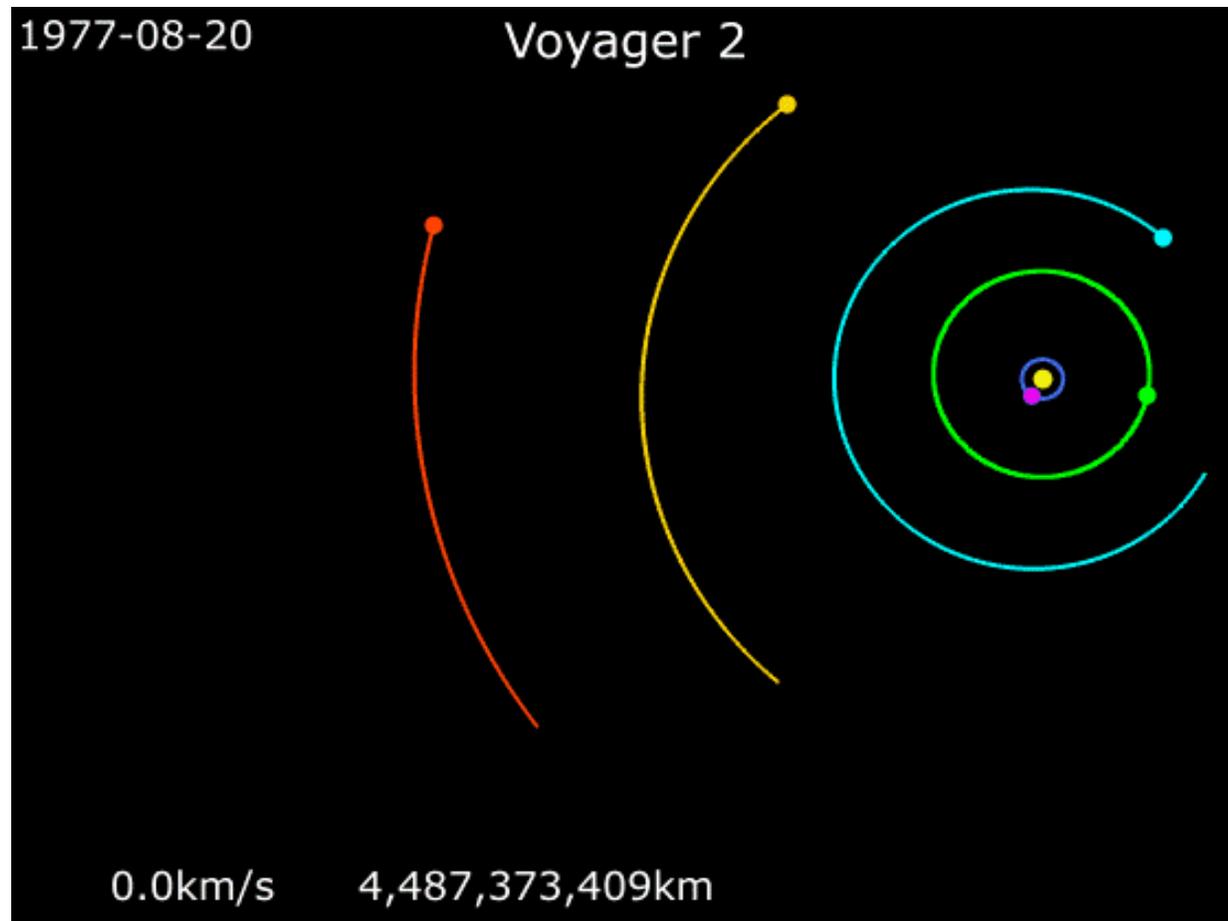
simulate: まねする，コピーする，
ふりをする，装う

(similar: 類似した)

他にもどのようなことが
できるか考えてみる

シミュレーションの活用例

- ▶ 宇宙探査機の動き（軌道）
- ▶ 右から地球(青), 火星(緑), 木星(水色), 土星(黄色), 天王星(赤)の重力の影響を受けた探査機ボイジャー(紫)の移動

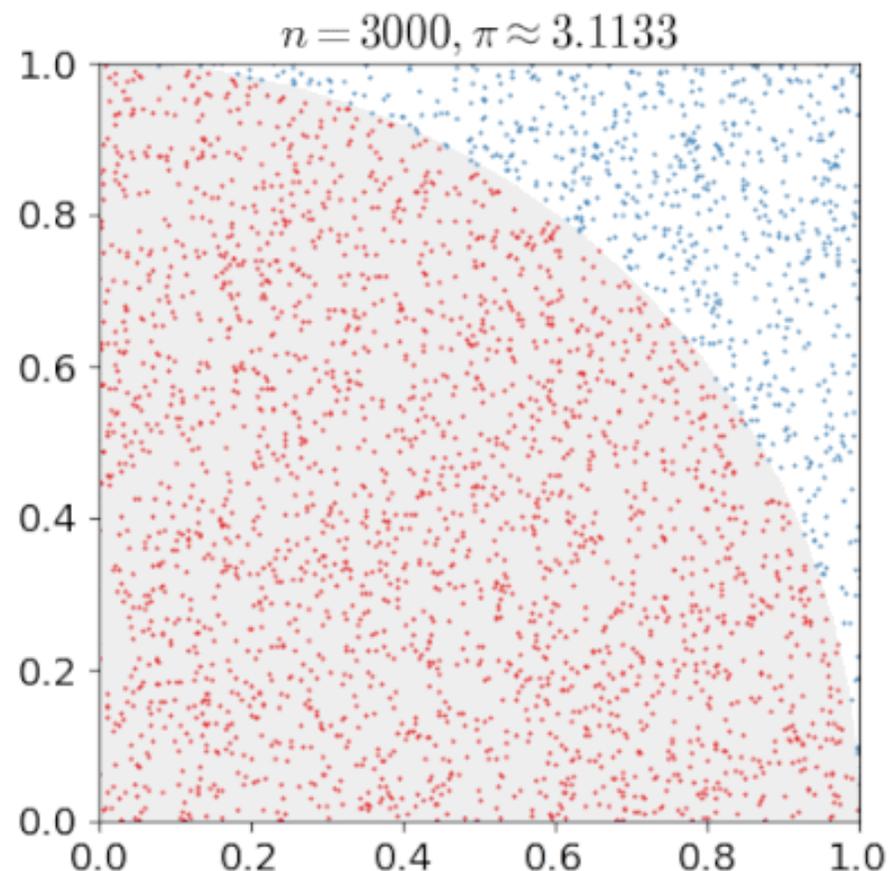


© NASA & JPL-Caltech

(<https://www.jpl.nasa.gov/edu/news/2018/12/18/then-there-were-two-voynager-2-reaches-interstellar-space/>)

シミュレーションの活用例

- ▶ **適当に**点を $(0,0) \sim (1,1)$ の範囲で作ったときに、半径1の円(単位円)の内側(赤)になるか、外側(青)になるか
- ▶ たくさん繰り返すと、全体に対する赤の数の比率は、 $\pi/4$ に近づく
- ▶ シミュレーションには乱数を使う
 - ▶ コンピュータでは疑似乱数
 - ▶ 乱数を使って確率的モデルをシミュレーションする方法を **モンテカルロ法** と言う



シミュレーションをやってみる
確定的な動的モデル

シミュレーションの準備

- ▶ Chromebook + Google Colab を使用 
 - ▶ ログインをする.
 - ▶ Web ブラウザ Chrome を開く.
 - ▶ 「Google Colab」 (Google Colaboartory) を開く.

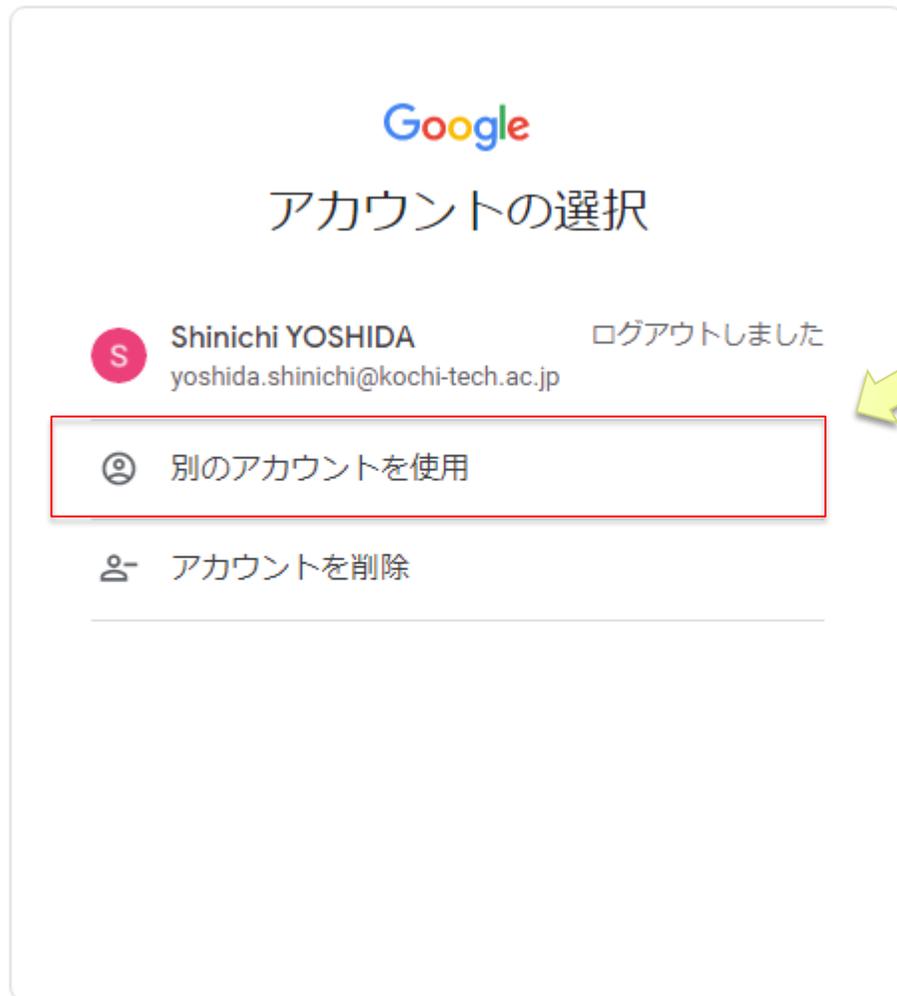
シミュレーションの準備

▶ Google colaboratory へアクセス



Google アカウントでの Google へのログイン

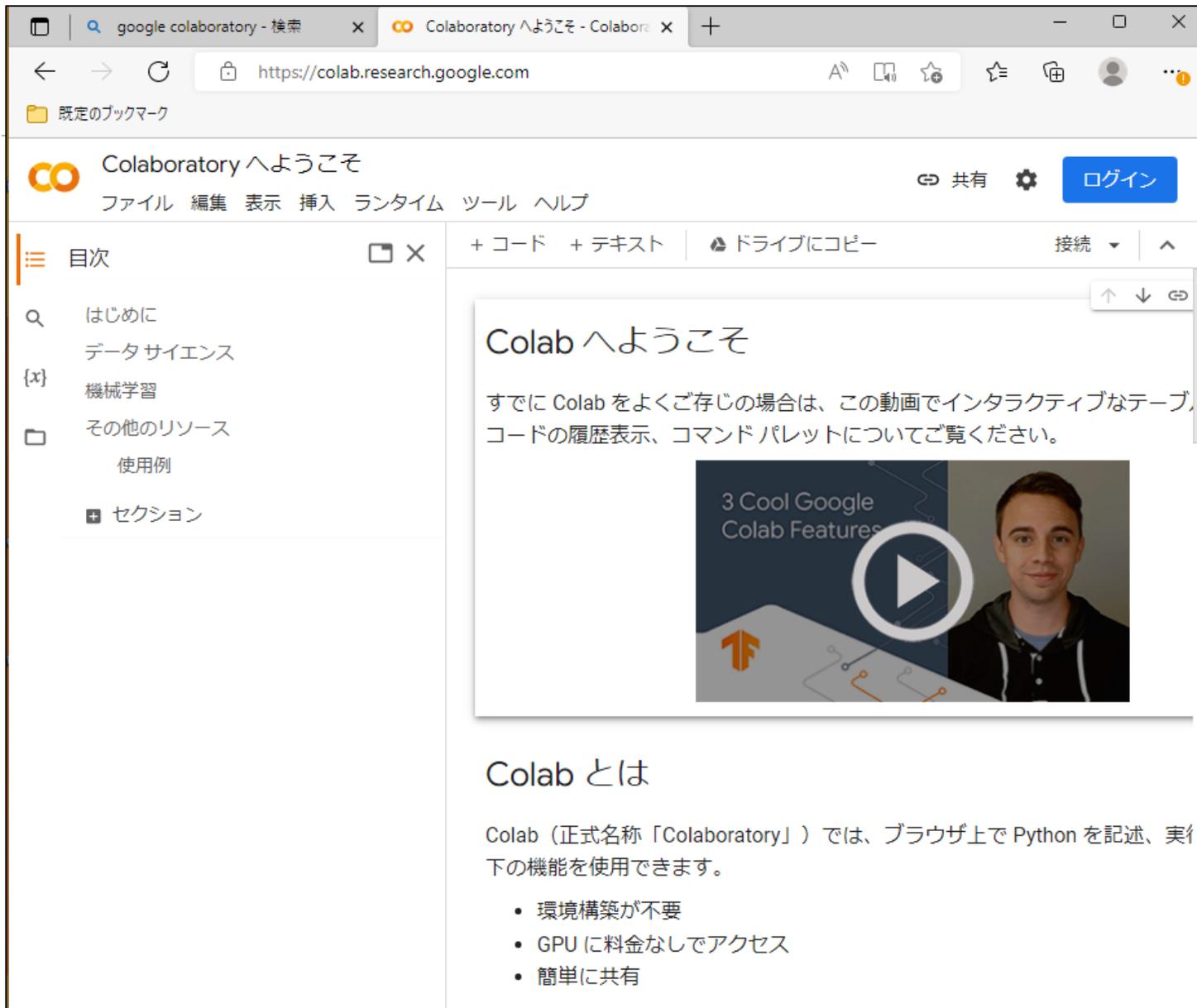
- ▶ 高校で使っている自分のアカウントで Google へログインしてください



Googleへログイン

▶ 右上「ログイン」

▶ 高校のChromebookで
使う Google の
アカウントでログイン



The screenshot shows the Google Colaboratory website interface. At the top right, there is a blue "ログイン" (Login) button. Below the header, there is a navigation menu with options like "ファイル", "編集", "表示", "挿入", "ランタイム", "ツール", and "ヘルプ". On the left side, there is a sidebar with a "目次" (Table of Contents) section containing links for "はじめに", "データサイエンス", "機械学習", "その他のリソース", "使用例", and "セクション". The main content area features a "Colab へようこそ" (Welcome to Colab) section with a video player showing "3 Cool Google Colab Features". Below this, there is a "Colab とは" (What is Colab) section with a brief description and a list of features.

Colab へようこそ

すでに Colab をよくご存じの場合は、この動画でインタラクティブなテーブル、コードの履歴表示、コマンドパレットについてご覧ください。

3 Cool Google Colab Features

Colab とは

Colab（正式名称「Colaboratory」）では、ブラウザ上で Python を記述、実行下の機能を使用できます。

- 環境構築が不要
- GPU に料金なしでアクセス
- 簡単に共有

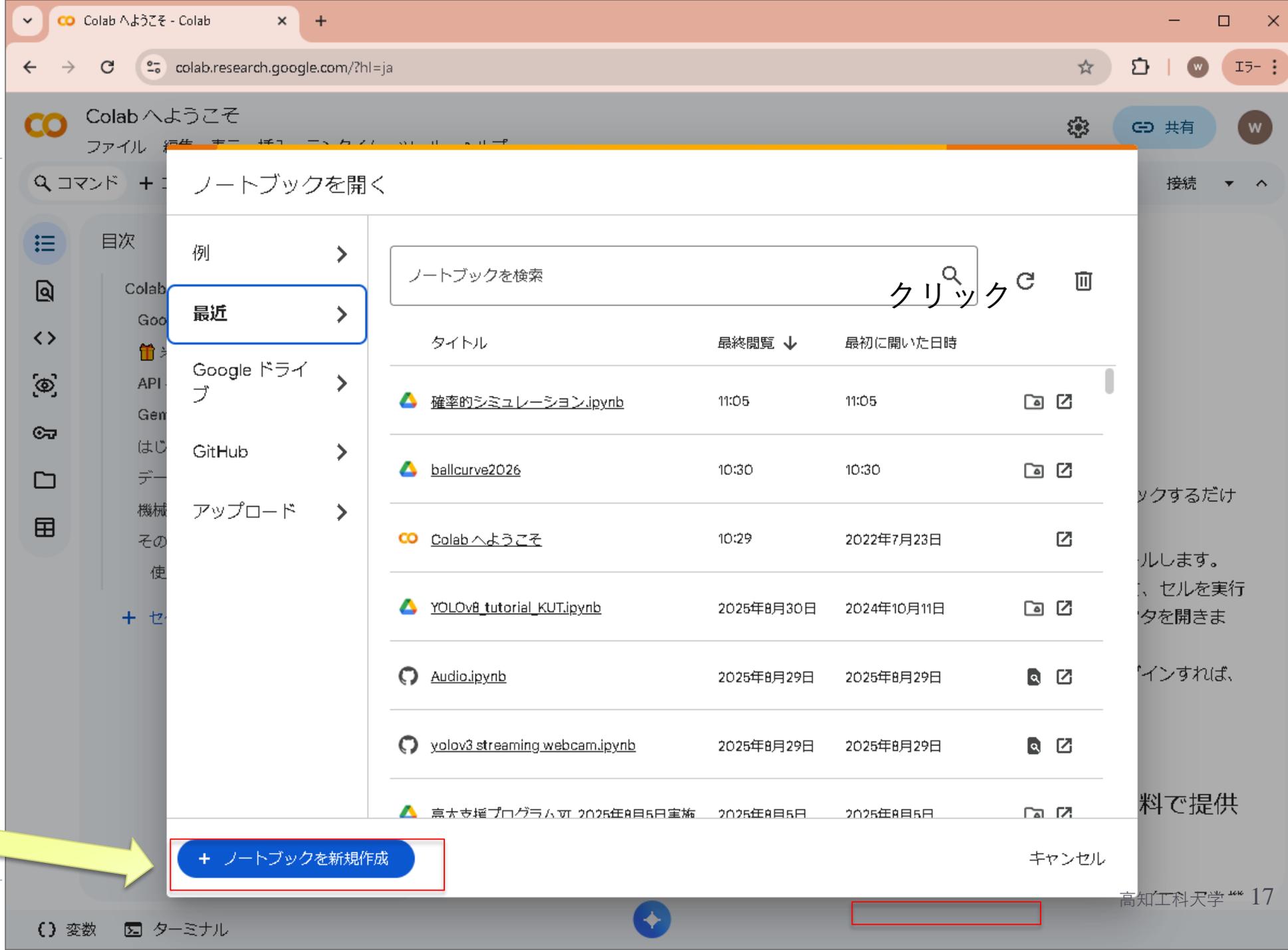
Googleへログイン

- ▶ 高校のChromebookで使う Google のアカウントでログイン



The image shows a screenshot of the Google login page in Japanese. At the top center is the Google logo. Below it, the text reads 'ログイン' (Login) and 'お客様の Google アカウントを使用' (Use your Google account). There is a text input field with the placeholder text 'メールアドレスまたは電話番号' (Email address or phone number). Below the input field, there is a link 'メールアドレスを忘れた場合' (If you forgot your email address). Further down, there is a paragraph of text: 'ご自分のパソコンでない場合は、シークレットブラウジングウィンドウを使用してログインしてください。' (If you are not on your own PC, please use the Secret Browsing window to log in.) followed by a link '詳細' (Details). At the bottom left, there is a link 'アカウントを作成' (Create account) and at the bottom right, a blue button labeled '次へ' (Next). At the very bottom of the page, there are links for '日本語' (Japanese), a dropdown arrow, 'ヘルプ' (Help), 'プライバシー' (Privacy), and '規約' (Terms).

▶ 「ノートブック」を新規作成

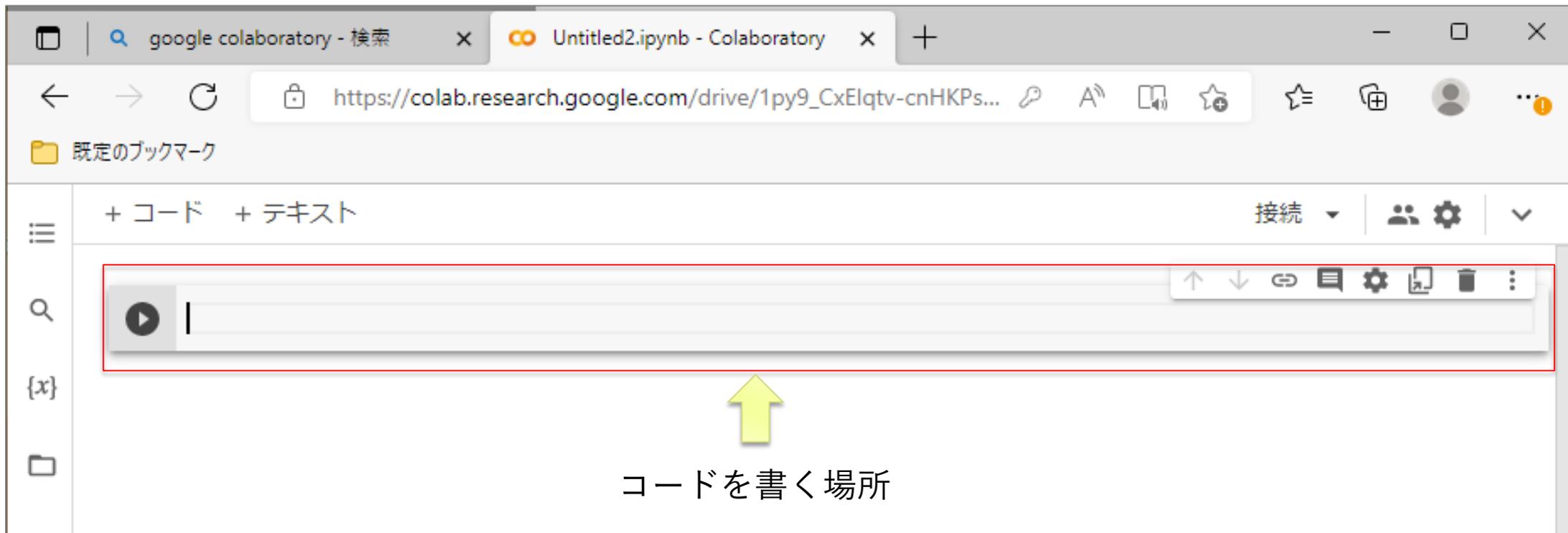


クリック

▶ 2026.3.13

Google Colaboratory ノートブック

- ▶ ノートブックは，文章(テキスト)，Pythonプログラム(コード)を自由に書き込み保存できる。
- ▶ コードを実行することもできる。



ブラウザでタブをもう一つ作成

- ▶ ブラウザでタブをもう一つ作成し，下記URLへアクセス
- ▶ https://github.com/yoshilab/kochi_hs_simulation

↓ タブの作成

github.com/yoshilab
の URL に行ってから、

`kochi_hs_simulation` Public

を探してクリック
しても良い

The screenshot shows a web browser with three tabs: 'google colab', 'Untitled2.ipynb - Colaboratory', and 'GitHub - yoshilab/kochi_hs_simulation'. The address bar shows the URL 'https://github.com/yoshilab/kochi_hs_simulation'. The page content includes a 'Sign up' button, the repository name 'yoshilab / kochi_hs_simulation', and navigation links for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Security', and 'Insights'. The file list shows three files: '1_ballcurve.py', '2_waiting_queue.py', and 'README.md'. A red box highlights the file list, and a yellow arrow points to the '+' icon in the browser tabs.

1時間目のプログラム →

2時間目のプログラム →

▶ 2026.3.13

Github アクセスがエラーが出る場合

- ▶ 検索「高知工科大 吉田研」で、「**知能情報学研究室**」をクリック



プログラムをGoogle Colab で実行する

- ▶ Github から1時間目のプログラム「1_ballcurve.py」をクリック

Python プログラムの一部

```
45 lines (40 sloc) | 2.54 KB
1 import matplotlib.pyplot as plt
2 import math
3
4 # シミュレーション関数
5 # speed:初速度km/h degree:角度 color:色 g:重力加速度
6 def simulation(speed,degree,color,g):
7     v0 = speed*1000/3600 # 初速度 m/s に変換
8     rad = math.radians(degree) # 投げ上げ角度をラジアンに変換
9     x = [0] # 開始時の x 座標
10    y = [0] # 開始時の y 座標
11    vx = [v0 * math.cos(rad)] # 開始時の速度 x 方向成分 (v0 cos θ)
12    vy = [v0 * math.sin(rad)] # 開始時の速度 y 方向成分 (v0 sin θ)
13    dt = 0.3 # シミュレーションの時間間隔Δt
14    i = 0 # 位置・速度データを格納するリストの番号(添字)
15    while y[i] >= 0: # y座標が0以上の間(空中にいる間)繰り返す
16        vx.append(vx[i]) # 次の時刻の速度(x成分)は vx |はそのまま変わらず(等速)
17        vy.append(vy[i] - g*dt) # 次の時刻の速度(y成分)は vy |重力加速度分減少
18        x.append(x[i] + vx[i]*dt) # 次の時刻の位置x|は速度vxのdx分だけ移動
19        y.append(y[i] + (vy[i] + vy[i+1])/2.0*dt)
20        # 次の時刻の位置y|は速度vyのdx分だけ移動
21        # vyは重力で変化するので、現時刻と次時刻の速度の平均
```

プログラムのコピー

▶ プログラム全体をコピーする

クリック
すべてコピーされる

45 lines (40 sloc) | 2.54 KB

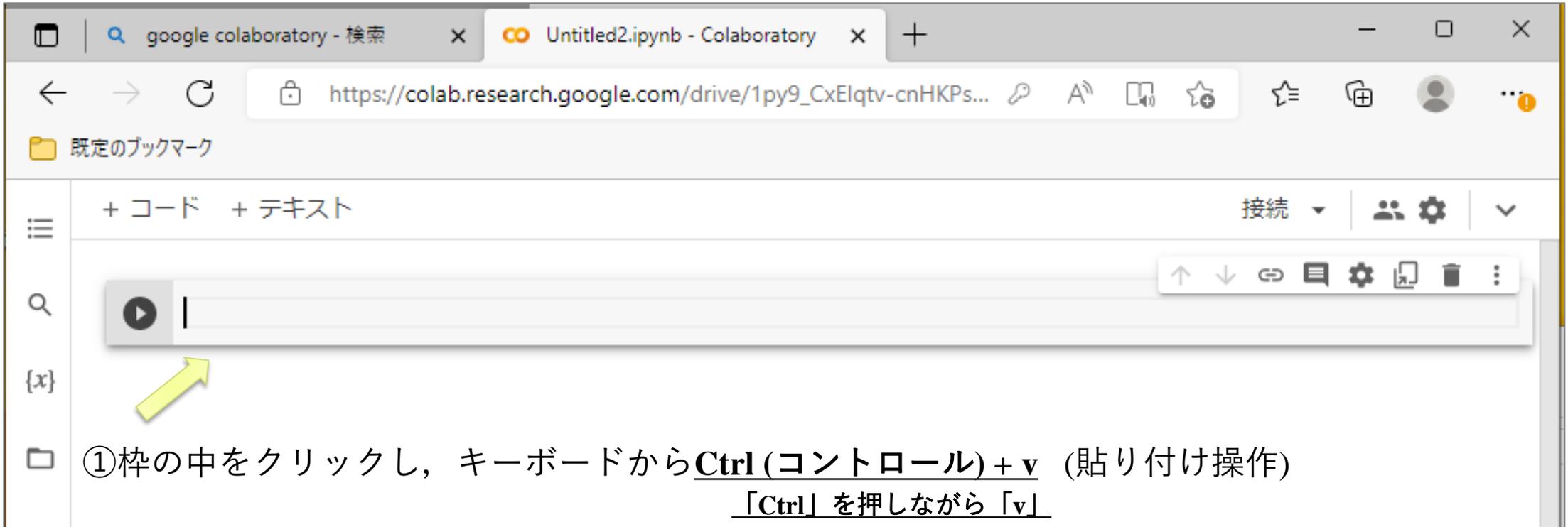
Raw

Blame



```
1 import matplotlib.pyplot as plt
2 import math
3
4 # シミュレーション関数
5 # speed:初速度km/h degree:角度 color:色 g:重力加速度
6 def simulation(speed,degree,color,g):
7     v0 = speed*1000/3600 # 初速度 m/s に変換
8     rad = math.radians(degree) # 投げ上げ角度をラジアンに変換
9     x = [0] # 開始時の x 座標
10    y = [0] # 開始時の y 座標
11    vx = [v0 * math.cos(rad)] # 開始時の速度 x 方向成分 (v0 cos θ)
```

Colab に貼り付け



The screenshot shows a web browser window with two tabs: "google colaboratory - 検索" and "Untitled2.ipynb - Colaboratory". The address bar shows the URL "https://colab.research.google.com/drive/1py9_CxEIqtv-cnHKPs...". The interface includes a sidebar with "既定のブックマーク", a top bar with "+ コード + テキスト" and "接続", and a main area with a code cell. A yellow arrow points to the play button icon in the code cell.

①枠の中をクリックし、キーボードからCtrl (コントロール) + v (貼り付け操作)
「Ctrl」を押しながら「v」

プログラム準備完了

- ▶ プログラムが全部コピーされる



```
CO Untitled1.ipynb ☆
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました

+ コード + テキスト
0秒
# speed:初速度km/h degree:角度 color:色 g:重力加速度
def simulation(speed,degree,color,g):
    v0 = speed*1000/3600 # 初速度 m/s に変換
    rad = math.radians(degree) # 投げ上げ角度をラジアンに変換
    x = [0]
    y = [0]
    vx = [v0 * math.cos(rad)]
    vy = [v0 * math.sin(rad)]
    t = 0 # 初期時刻
    dt = 0.3 # 時間間隔
    i = 0
    while True:
        vx.append(vx[i])
        vy.append(vy[i] - g*dt)
        x.append(x[i] + vx[i]*dt)
        y.append(y[i] + (vy[i] + vy[i+1])/2.0*dt)
        if y[i] < 0:
            break
        i = i + 1
    label_text = "v=" + str(speed) + "km/h, " + "angle=" + str(degree) + ", dist"
    plt.scatter(x, y, color=color, label=label_text)

    return

# main 処理
# ここからプログラム開始

# 投げ上げ角度を変えながらシミュレーションを行い, 色を変えてプロット
simulation(speed=162, degree=80, color="red", g=9.8) # 初速v0, 角度degree, 重力
simulation(speed=162, degree=60, color="purple", g=9.8) # 初速v0, 角度degree, 重
simulation(speed=162, degree=30, color="gray", g=9.8) # 初速v0, 角度degree, 重
simulation(speed=162, degree=10, color="orange", g=9.8) # 初速v0, 角度degree, 重

# グラフ描画
plt.gca().set_aspect("equal", adjustable="box") #アスペクト (縦横比) を1:1に
plt.xlim(0,200) # x軸は0~200m
```

プログラムを実行

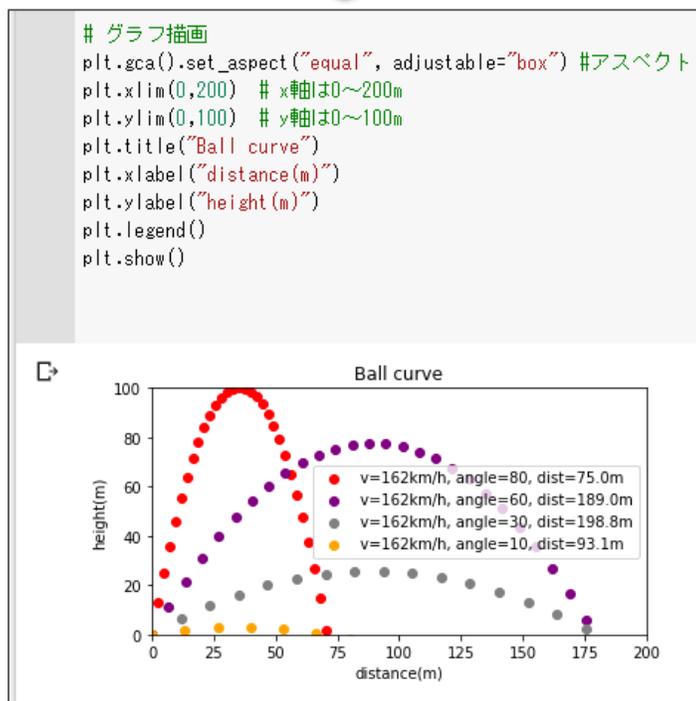
▶ 実行ボタンをクリック

ボタンクリック



実行に移る

プログラムの下にグラフが表示される



- 同じ結果になりましたか
- 隣の人とはどうですか？

```
Untitled1.ipynb ☆
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました

+ コード + テキスト
# speed:初速度km/h degree:角度 color:色 g:重力加速度
def simulation(speed,degree,color,g):
    v0 = speed*1000/3600 # 初速度 m/s に変換
    rad = math.radians(degree) # 投げ上げ角度をラジアンに変換
    x = [0]
    y = [0]
    vx = [v0 * math.cos(rad)]
    vy = [v0 * math.sin(rad)]
    t = 0 # 初期時刻
    dt = 0.3 # 時間間隔
    i = 0
    while True:
        vx.append(vx[i])
        vy.append(vy[i] - g*dt)
        x.append(x[i] + vx[i]*dt)
        y.append(y[i] + (vy[i] + vy[i+1])/2.0*dt)
        if y[i] < 0:
            break
        i = i + 1
    label_text = "v=" + str(speed) + "km/h, " + "angle=" + str(degree) + ", dist"
    plt.scatter(x, y, color=color, label=label_text)

    return

# main 処理
# ここからプログラム開始

# 投げ上げ角度を変えながらシミュレーションを行い、色を変えてプロット
simulation(speed=162, degree=80, color="red", g=9.8) # 初速v0, 角度degree, 重力
simulation(speed=162, degree=60, color="purple", g=9.8) # 初速v0, 角度degree, 重
simulation(speed=162, degree=30, color="gray", g=9.8) # 初速v0, 角度degree, 重
simulation(speed=162, degree=10, color="orange", g=9.8) # 初速v0, 角度degree, 重

# グラフ描画
plt.gca().set_aspect("equal", adjustable="box") #アスペクト (縦横比) を1:1に
plt.xlim(0,200) # x軸は0~200m
```

シミュレーション課題

- ▶ 初速度162キロのまま，角度をいろいろ変えたとき最も遠くに飛ぶ角度は何度か？
- ▶ 微小時間 $dt = 0.3$ 秒ごとにシミュレーションを行っているが，これを0.1, 0.01 と変化させると距離は変わるか？
 - ▶ 正しい距離はどの値？（ヒント：計算誤差）
- ▶ 重力加速度 g を $1/6$ の値にすると，162キロで投げたボールは最大どこまで飛ぶか？
 - ▶ （月の重力）

ここまでのまとめ

- ▶ 動的モデル（確定的）のシミュレーション
- ▶ 投げ上げたボールの動きを物理学での運動の方程式でモデル化
- ▶ コンピュータシミュレーション
 - ▶ 微小時間 dt 秒ごとに、速度や位置を順に計算（逐次的）
 - ▶ 角度 θ , 初速度 v_0 , 重力加速度 g をいろいろ変えて実行（パラメータ）

シミュレーションをやってみる
確率的な動的モデル

2時間目のプログラムをコピー

- ▶ ブラウザでタブをもう一つ作成し，下記URLへアクセス
- ▶ https://github.com/yoshilab/kochi_hs_simulation

↓ タブの作成

The screenshot shows a web browser with multiple tabs. The active tab is 'GitHub - yoshilab/kochi_hs_simulation'. The browser address bar shows the URL 'https://github.com/yoshilab/kochi_hs_simulation'. The page content displays the repository structure for 'yoshilab / kochi_hs_simulation'. The file list includes '1_ballcurve.py', '2_waiting_queue.py', and 'README.md'. The file '2_waiting_queue.py' is highlighted with a red box. The right sidebar shows repository statistics: 0 stars, 2 watching, and 0 forks.

2_waiting_queue.py

2時間目のプログラム →

▶ 2026.3.13

プログラムのコピー

51 lines (45 sloc) | 2 KB

Raw Blame    

```
1 # coding: utf-8
2
3 import math
4 import random
5 import matplotlib.pyplot as plt
6
7 time_c = 5.0      # 次の客(customer)が来るまでの平均時間(分)
8 time_s = 1.0     # 平均的なレジのサービス時間(service)(分)
9
10 t = 0
11 tc = 0
12 n = 0
13
14 t_all=[ ]      # 行列の人数が変化する時刻
15 n_all=[ ]     # 行列の人数
```



コードのセルを追加

- ▶ コードのセルを追加 (+コード)

クリック



The screenshot shows a Jupyter Notebook titled 'Untitled1.ipynb'. The top menu bar includes 'ファイル', '編集', '表示', '挿入', 'ランタイム', 'ツール', and 'ヘルプ'. Below the menu, there are two buttons: '+コード' (highlighted with a red box) and '+テキスト'. A yellow arrow points to the '+コード' button from the text 'クリック'. Below the buttons is a search bar and a toolbar with icons for home, search, and a code cell icon. The main content area displays a plot titled 'Ball curve' with the following legend:

Color	Velocity (v)	Angle	Distance (dist)
Red	150 km/h	70°	115.4 m
Purple	150 km/h	60°	156.3 m
Grey	150 km/h	45°	185.6 m
Yellow	150 km/h	30°	162.4 m

The plot shows height (m) on the y-axis (0 to 100) and distance (m) on the x-axis (0 to 200). Below the plot, a yellow arrow points to a play button icon, with the text '追加される' (to be added) next to it.

追加される



確率的モデルのプログラム

- ▶ Ctrl + v で
貼り付け

```
import math
import random
import matplotlib.pyplot as plt

time_c = 5.0      # 次の客(customer)が来るまでの平均時間(分)
time_s = 1.0      # 平均的なレジのサービス時間(service)(分)

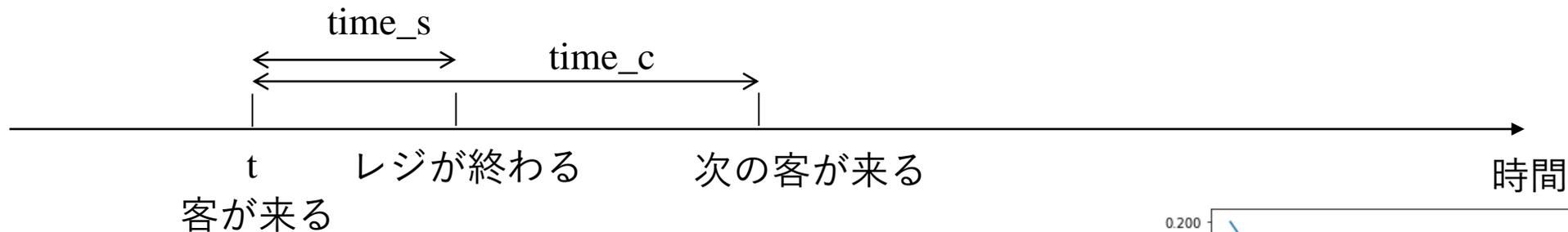
t = 0
tc = 0
n = 0

t_all=[ ] # 行列の人数が変化する時刻
n_all=[ ] # 行列の人数

while t < 240: # 240分(4時間)のシミュレーション
    n_prev = n
    if n == 0: # もしレジの客が0人なら
        x = -time_c * math.log(1-random.random()) # 次の客が来るまでの時間
        t = tc
```

確率的な動的モデルのプログラムの説明

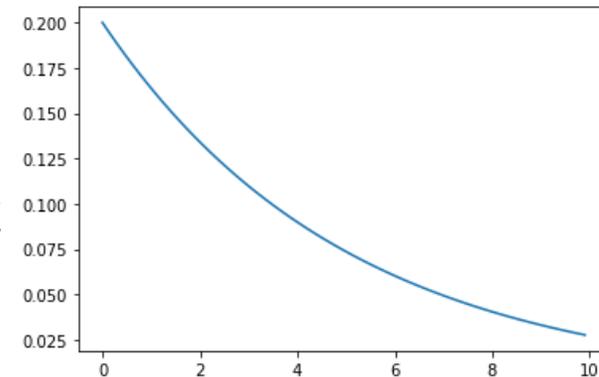
- ▶ コンビニエンスストアのレジに並ぶ客の人数
- ▶ 待ち行列理論：確率的に人が来店するモデル
- ▶ レジに客(customer)が来る間隔：平均して time_c 分
- ▶ レジでのサービス(service)の時間：平均して time_s 分



time_c : 1分あたり $(1/\text{time_c})$ 人来る指数分布

来る確率 $p = \lambda e^{-\lambda x}$ 来る確率

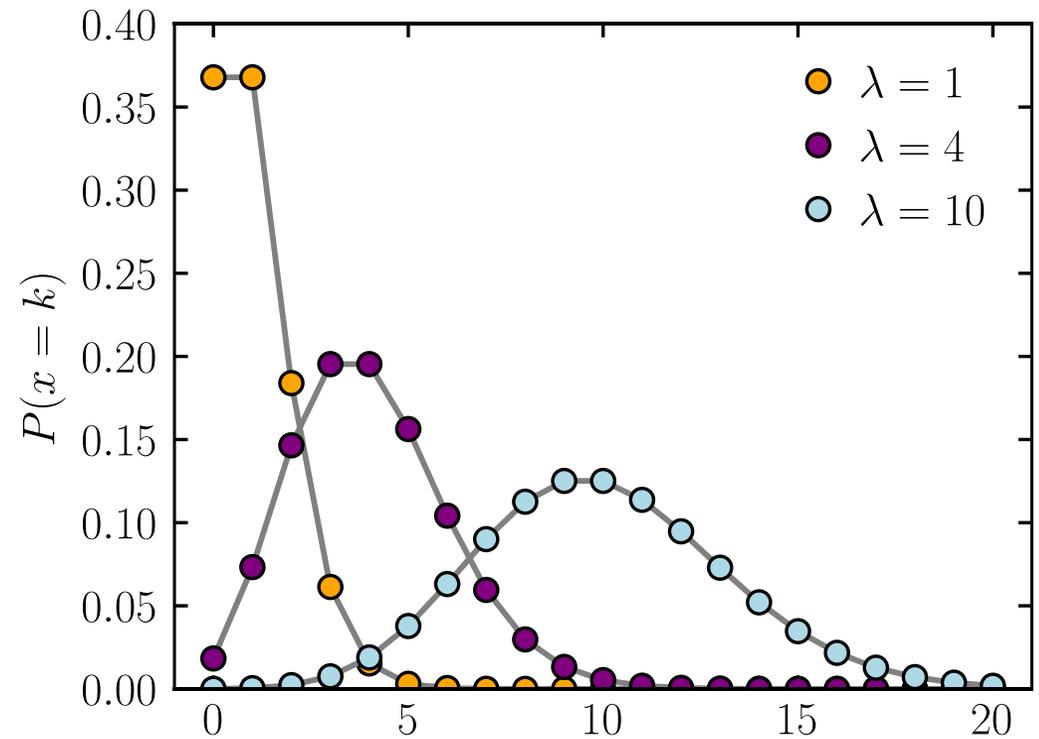
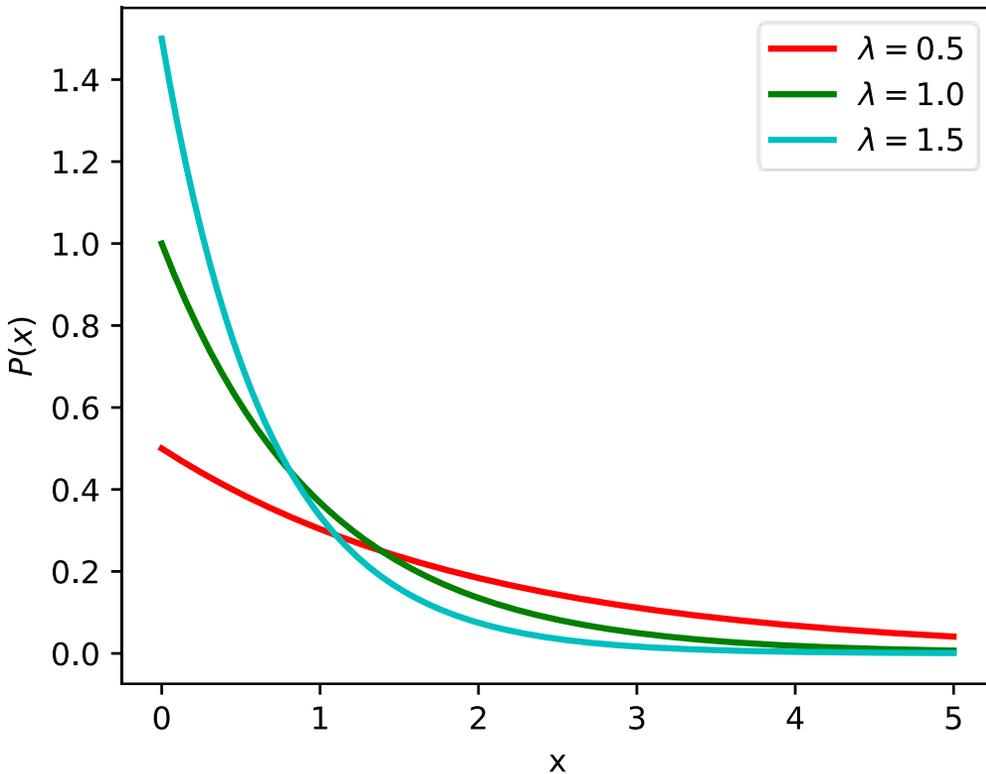
ただし $\lambda = \frac{1}{\text{time_c}}$



(番外編) 待ち行列の理論：指数分布とポアソン分布

指数分布：平均到着間隔 $1/\lambda$ の確率分布

ポアソン分布：単位時間中に平均 λ 回の事象が発生する確率分布



例：次のお客さん来るまでの時間

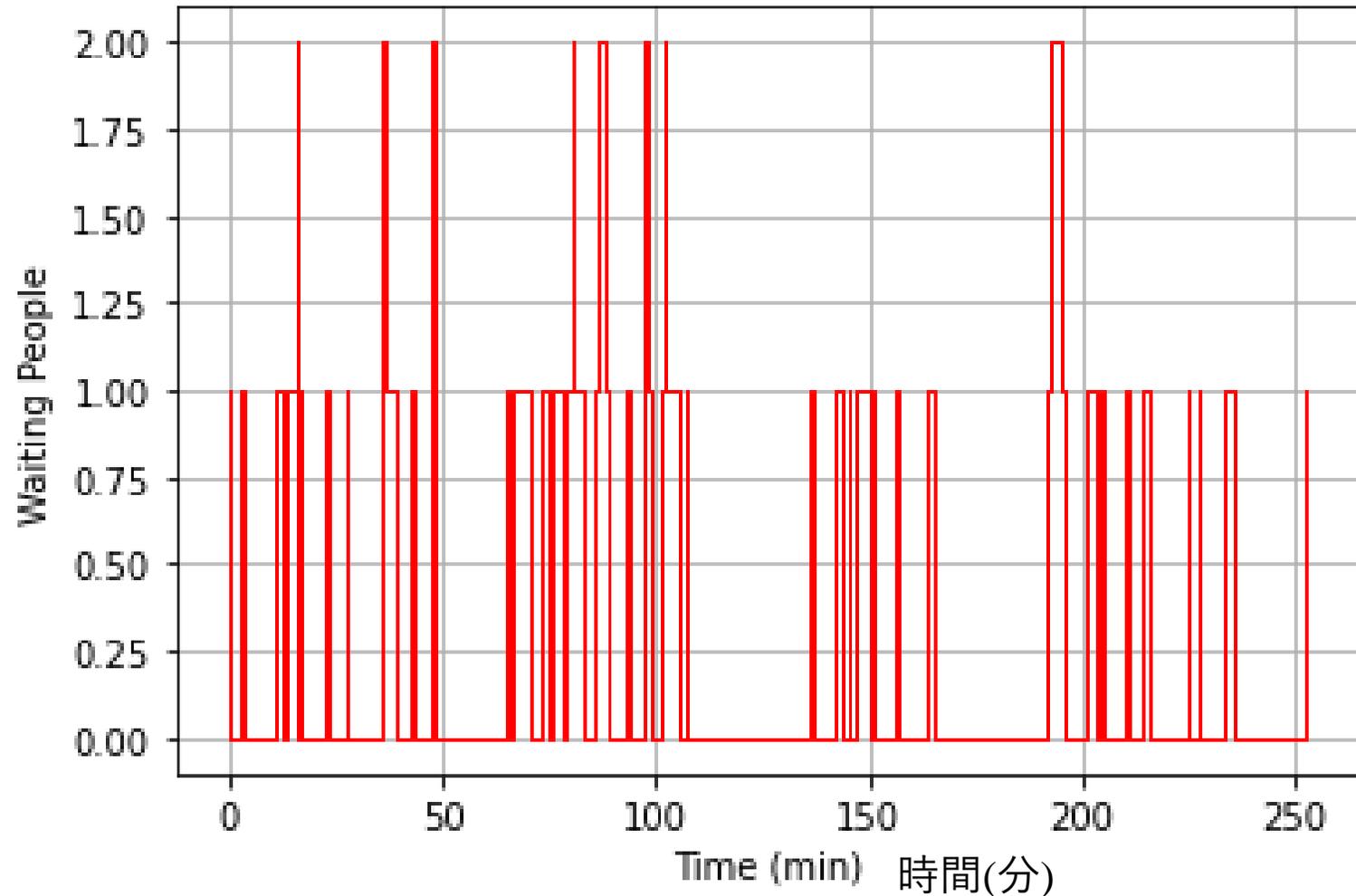
例：一定時間 (k 1時間等) に来店する客数

待ち行列のシミュレーション結果

▶ 実行結果

- ▶ 私と同じ結果になりましたか？
- ▶ 隣の人とはどうですか？

レジで待っている人の数



確率的な動的モデルのシミュレーション

- ▶ 乱数が使われる
- ▶ 実行するたびに結果が異なる

- ▶ パラメータを変えてみる
- ▶ 客がレジに来る間隔 time_c を増やす, または減らす
- ▶ レジの処理時間 time_s を増やす, または減らす

ここまでのまとめ

- ▶ 動的モデル（確率的）のシミュレーション
- ▶ コンビニのレジに並ぶ客の数
- ▶ 待ち行列理論

- ▶ コンピュータシミュレーション
 - ▶ 乱数を使う
 - ▶ 毎回異なる結果が得られる

シミュレーション課題

- ▶ $\text{time}_c = 5, \text{time}_s = 1$ で最大どの程度まで並ぶか
 - ▶ $\text{time}_c = 2$ となるとどうか

- ▶ $\text{time}_s \geq \text{time}_c$ の場合, 待ち行列はどうか

シミュレーションのまとめ

- ▶ モデル化
 - ▶ 現実世界の対象を模倣する
 - ▶ 静的モデル, 動的モデル, 確定的, 確率的
- ▶ シミュレーション
 - ▶ 現実世界のことを, 模擬的に実行する
 - ▶ コンピュータシミュレーション
- ▶ 動的なモデルのシミュレーション
 - ▶ 確定的: 物理学などの方程式に基づくモデル
 - ▶ 確率的: 確率に基づいて乱数を使ってシミュレーションする
- ▶ 現実を模擬するが, 全く同じになるか
 - ▶ 計算の精密さ, 計算誤差は必ず発生する

この資料

▶ この資料のPDFファイルは、

にあります。

ふりかえり

- ▶ モデルとシミュレーションとは
 - ▶ 様々なモデルやシミュレーションの例
- ▶ 動的モデルのシミュレーションの2種類
 - ▶ どのような違いがあるか
 - ▶ どのような応用が考えられるか

- ▶ 発展
 - ▶ 「シミュレーション」は現代の科学研究において重要な要素
 - ▶ 物理・化学・生物・・・
 - ▶ 理論科学，実験科学に続く，第3の科学と呼ぶ人もいる
 - （最近では，データサイエンスを第4の科学と呼ぶ例もある）