2023/09/05

Pythonを用いたモデルロケット飛行の数値解析

日大ロケット研究会　代表　内藤正樹

はじめに

- 1年生, 機械系学科以外の方には少し難しいかもしれません.
- Open Rocketは内部で計算していることがわからないですね.
- 自分でシミュレーションをすることで, ロケット工学の理解を深めましょう.
- ソースコードなどはGitHubにアップロードしてあります.
  https://github.com/NUROP-2023/Python-model-rocket-simulation

質点モデルと剛体モデル

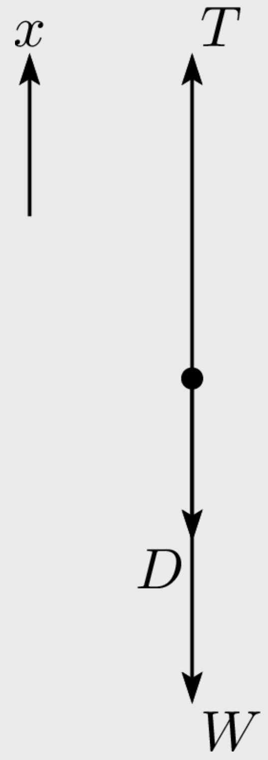
- 質点モデルでは, ロケットを**大きさのない点**として見なす.
- つまり, **モーメント**と**回転**を**考慮しない**.

- 一方剛体モデルは, 大きさを考え, 回転も取り扱う.

- 一般に, 剛体モデルは**計算が複雑**になるため, 今回は質点モデルについて考える.

1次元1自由度シミュレーション

- 下図のように, 並進1自由度(上下運動のみ)に簡略化して考える.



- T: 推力 Thrust
- D: 抗力 Drag
- W: 重力 Weight

- 運動方程式は $m\dfrac{d^2x}{dt^2} = T - D - W$ となる.

推力Tの計算

- 推力は実際には右図のように変化.
- 簡単のため, 推力は一定と考える.
- 燃焼時間t_b, 平均推力T_aとすると

$$T(t) = \begin{cases} T_a & (t \leq t_b) \\ 0 & (t > t_b) \end{cases}$$

- なお, 平均推力はトータルインパルスIと燃焼時間t_bから

$$T_a = \frac{I}{t_b}$$

本当はスラストカーブの画像
（当日のパワポには入れてますが, 著作権
の観点から消してあります）

引用元

https://estesrockets.com/products/a8-3-engines

抗力Dの計算

- 抗力Dの大きさは次式で与えられる.

$$D = \frac{1}{2} C_D \rho v^2 S$$

- ここで, C_Dは抗力係数, ρは大気密度, vは速度, Sは代表面積.
- C_Dはロケットによって異なり, 風洞実験または計算で求める.
- ρは標準大気では1.225 $\mathrm{kg/m^3}$.
- Sは機体を機軸直交方向の面で切断した最大断面積を用いる.
- 抗力は速度ベクトルと逆向きに作用.
- $v > 0$なら$D < 0$, $v < 0$なら$D > 0$.

重力Wの計算

- 重力Wは

$$W = m_m g$$

- m_mは打ち上げ平均質量.
- 機体質量m_b, エンジン質量m_e, 推進薬量m_pとすると

$$m_m = m_b + m_e - \frac{m_p}{2}$$

- gは重力加速度で, 9.80665 $\mathrm{m/s^2}$.

運動方程式まとめ

➢ 以上の導出から運動方程式は,

$$m\frac{d^2x}{dt^2} = T - D - W$$

常に速度と逆向き

$$T = \begin{cases} T_a, & t \leq t_b \\ 0, & t > t_b \end{cases}, \qquad D = \frac{1}{2}C_D\rho v|v|S, \qquad W = m_m g$$

➢ これは非線形2階常微分方程式で, 手計算で解くのは無理.

➢ ということで, Pythonでプログラムする.

Pythonで常微分方程式を解く

- scipy.integrateのodeintを使って常微分方程式を解く
- 1階連立微分方程式を解くための関数
- $\frac{d\boldsymbol{y}}{dt} = f(\boldsymbol{y}, t, \dots)$の形式で与える.
- 前ページの式を以下のように変形

$$\begin{cases} \dfrac{dx}{dt} = v \\ \dfrac{dv}{dt} = \dfrac{T - D - W}{m_m} \end{cases}$$

シミュレーションするロケット

- NUROPのチーム「いにしゃんず」が昨年度種子島ロケットコンテストに出場した機体でシミュレーションする.

直径 [mm]	56
全長 [mm]	554
エンジン抜き質量 [g]	124
$C_D(AOA = 0\mathrm{deg})$ [-]	0.55
$C_D(AOA = 90\mathrm{deg})$ [-]	23

Pythonで実装する

- 定数を最初に書いておく.
- 単位を間違えないように工夫.
- 今回のプログラムは, 初心者向けに, わかりやすさ重視のものです.
- 本来は, 設定ファイルをyamlなりjasonなりで作った方が良い.

```python
import math
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint
```

```python
# エンジン諸元
# C11-3の場合
TOTAL_IMPULSE = 8.8            # トータルインパルス [N s]
BURN_TIME = 0.8                # 燃焼時間 [s]
PROP_MASS = 12 / 1000          # 推進薬量 [kg]
ENGINE_MASS = 35.3 / 1000      # エンジン質量[kg]

# 機体諸元
# いにしゃんず種コン2023の場合
DIAMETER = 56 / 1000           # 機体直径 [m]
BODY_MASS = 124 / 1000         # エンジン抜き機体質量 [kg]
C_D = 0.55                     # 抗力係数 [-]

# 物理定数
RHO = 1.225                    # 大気密度 [kg/m^3]
GRAVITY = 9.80665              # 重力加速度 [m/s^2]

# 予め計算
AVERAGE_THRUST = TOTAL_IMPULSE / BURN_TIME                        # 平均推力 [N]
AREA = math.pi * (DIAMETER)**2 / 4                                # 断面積 [m^2]
AVERAGE_MASS = BODY_MASS + ENGINE_MASS - PROP_MASS / 2            # 打ち上げ平均質量[kg]
W = AVERAGE_MASS * GRAVITY                                        # 重力 [N]

# 解析時間
ANALYSIS_TIME = 10                 # 解析時間 [s]
DT = 0.01                          # 時間刻み [s]
DIV = int(ANALYSIS_TIME / DT)      # 分割数

# ファイル保存名
FIGURE_NAME = "いにしゃんず種コン2023"
```

Pythonで実装する

- 運動方程式は関数として分ける.

```python
def eom(X, t):
    x, v = X

    thrust = 0
    if t <= BURN_TIME:
        thrust = AVERAGE_THRUST

    drag = (RHO * v * abs(v) * AREA * C_D) / 2

    a = (thrust - drag - W) / AVERAGE_MASS

    return [v, a]
```

Pythonで実装する

- 数値積分する部分も関数として分ける.

```python
def simulation():
    x_0 = 0
    v_0 = 0
    X_0 = [x_0, v_0]

    t = np.linspace(0, ANALYSIS_TIME, DIV)
    sol = odeint(eom, X_0, t)

    return t, sol
```

Pythonで実装する

- グラフを描く関数と保存する関数.
- Cと違い, グラフまで1つのプログラムで書けるのがPythonの魅力.

```python
def plot_graph(t, x, v):
    fig = plt.figure(tight_layout=True)
    ax = fig.add_subplot()

    ax.plot(t, x, label=r"$x$")
    ax.plot(t, v, label=r"$v$")

    ax.set_xlabel(r'$t$ [s]')
    ax.set_ylabel(r'$x$ [m], $v$ [m/s]')
    ax.grid(color='black', linestyle='dotted')
    ax.legend()

    plt.show()

    return fig
```

```python
def save_fig(fig):
    fig.savefig(f"{FIGURE_NAME}.png", dpi=300)
```

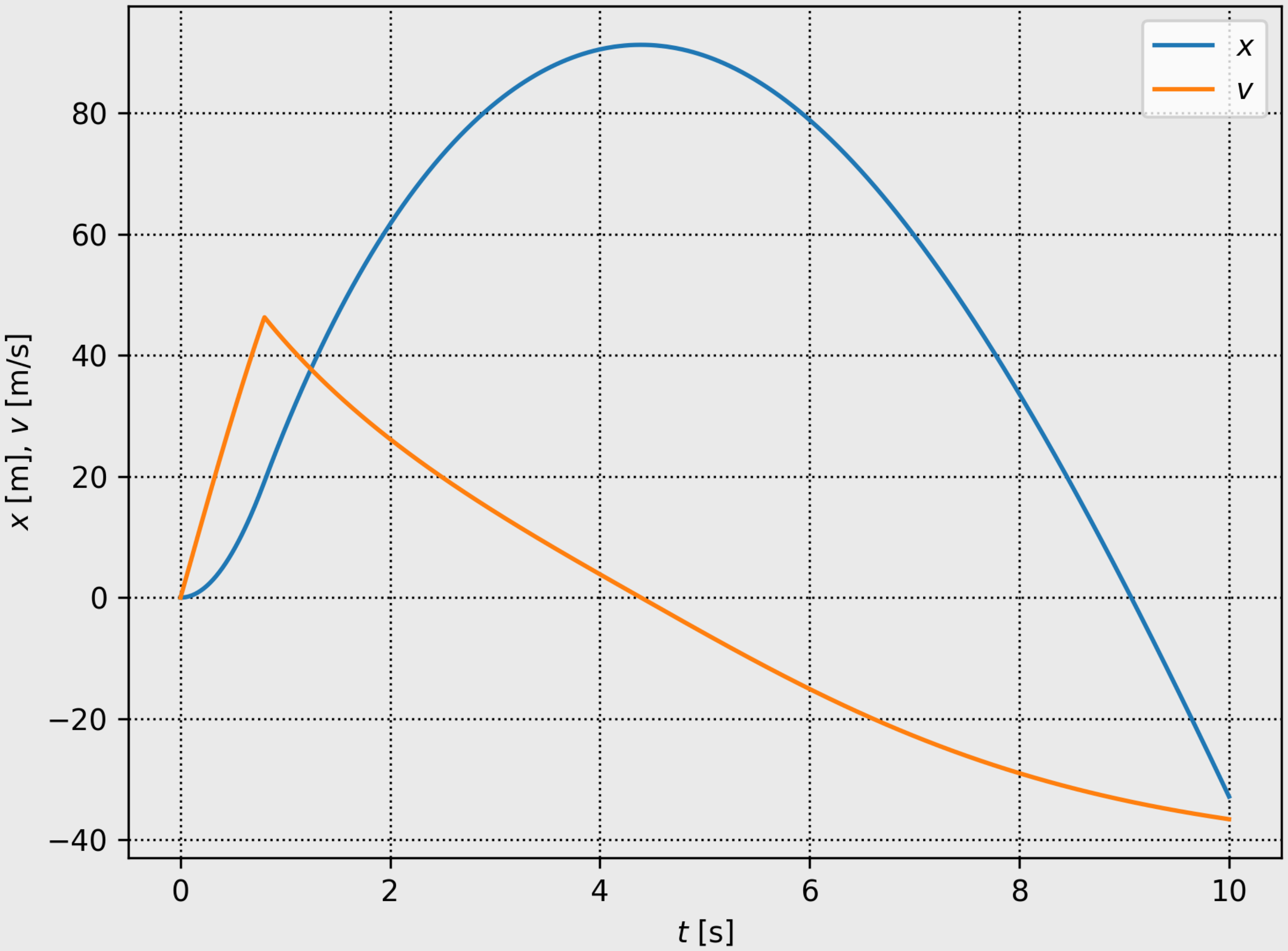
Pythonで実装する

- main部分は, 関数を呼び出している.

```python
if __name__ == '__main__':
    t, sol = simulation()

    x_sol = sol[:, 0]
    v_sol = sol[:, 1]

    max_altitude = max(x_sol)
    max_velocity = max(v_sol)
    print(f"max altitude: {max_altitude:.1f} [m]")
    print(f"max velocity: {max_velocity:.1f} [m/s]")

    fig = plot_graph(t, x_sol, v_sol)
    save_fig(fig)
```

実行結果

- 実行結果は右図.
- パラシュートは開いていないので, 弾道飛行.

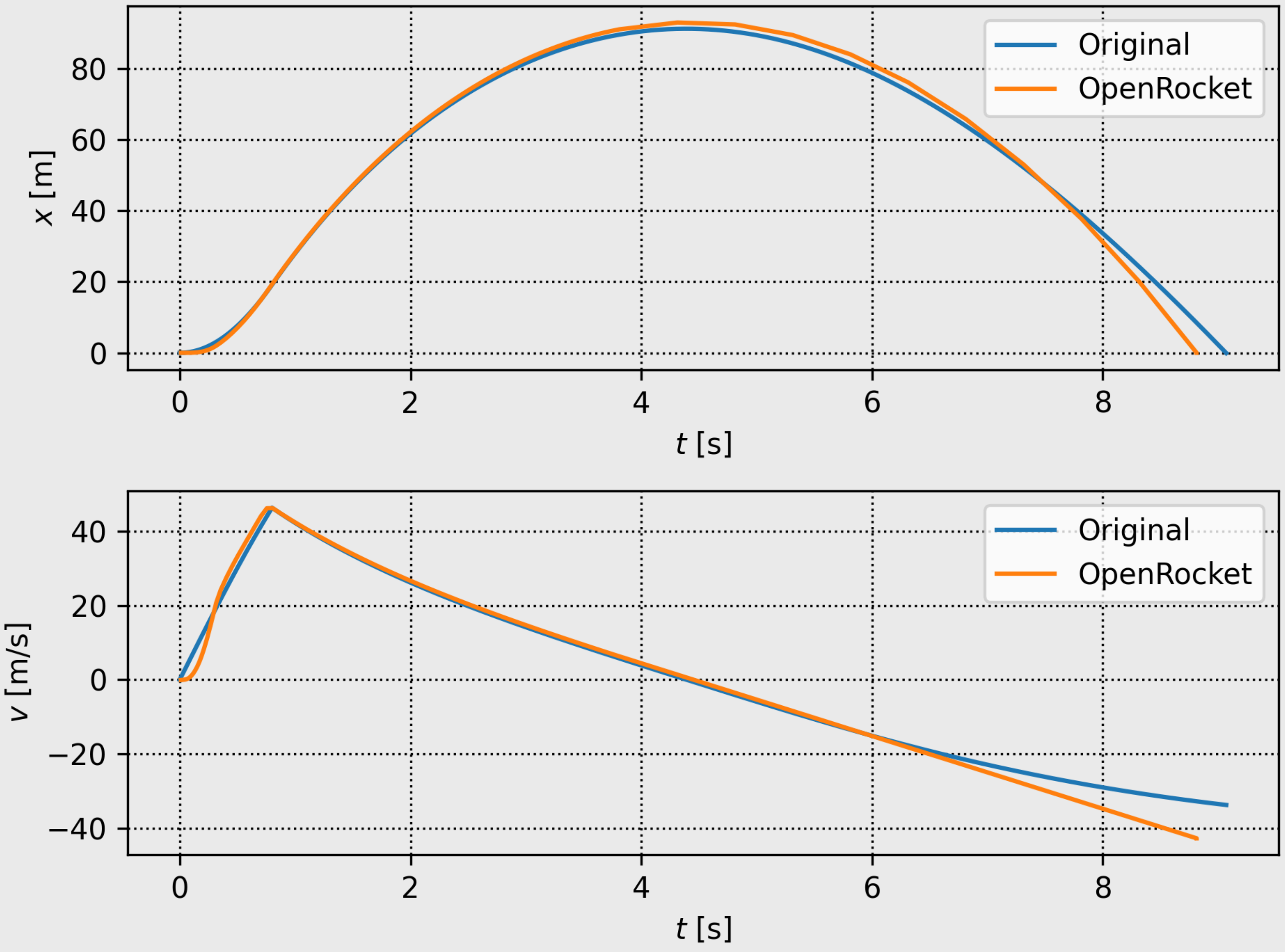


Open Rocketと比較

- いくつかの近似をしたが, かなり高精度で一致している.

	最高高度 [m]	最大速度 [m/s]
自作シミュレーション	**91.3**	**46.3**
Open Rocket	**93.0**	**46.2**
相対誤差 （Open Rocketを真の値とする）	**-1.83%**	**0.22%**

Open Rocketと比較



今回のシミュレーションの改善点

- 実際のロケットは剛体であり, 回転するため, 風などの外乱があれば垂直上昇するとは限らない.
- 大気密度と重力加速度の変化を考慮していない. ただ, 100m程度なら, それぞれ誤差は0.9%と0.003%.
- 平均質量･平均推力を用いているが, 実際には時間変化する.
- 地球の自転は考慮していない.

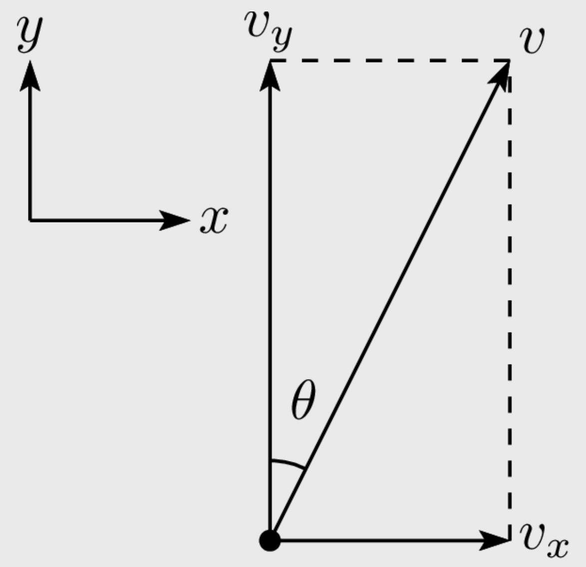
シミュレーションの利用

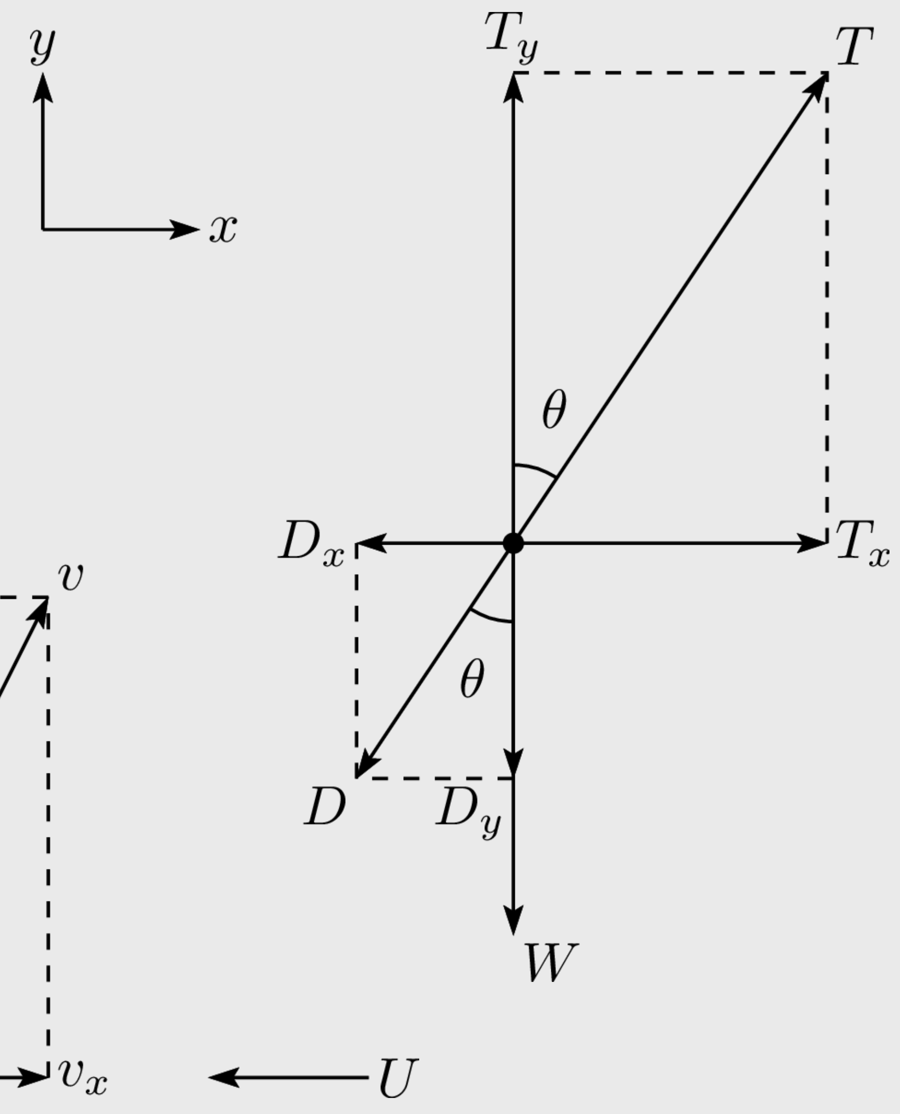
- 1次元の質点モデルでも, ロケットの到達高度や速度が把握できる.
- 正確ではないかも知れないが, “傾向“がわかる.
- これを利用すれば
  - より高く飛ぶロケットの設計
  - ランチロッドクリアースピードを満たす条件の確認
- など, 条件を変化させた場合の変化を見られる.

2次元2自由度系シミュレーション

- 斜めに打ち上げることを考える.
- 右図のようになる.
- 運動方程式は

$$
\begin{cases}
m\dfrac{d^2x}{dt^2} = T_x - D_x \\
m\dfrac{d^2y}{dt^2} = T_y - D_y - W
\end{cases}
$$





2次元2自由度系シミュレーション

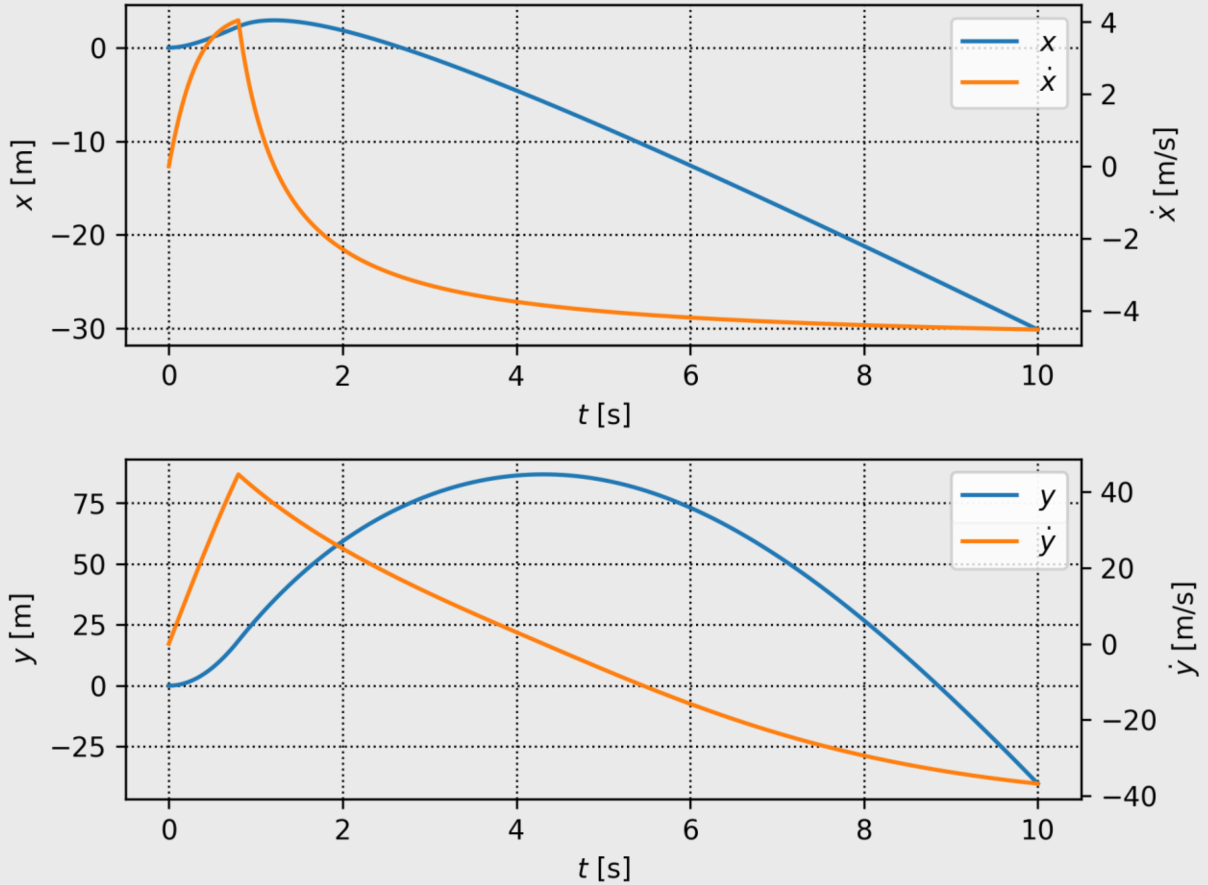
- 式変形すると

$$\begin{cases} m\dfrac{d^2x}{dt^2} = T(t)\sin\theta - \dfrac{1}{2}C_{D_x}\rho\left(\dfrac{dx}{dt}+U\right)\cdot\left|\dfrac{dx}{dt}+U\right|S_x \\ m\dfrac{d^2y}{dt^2} = T(t)\cos\theta - \dfrac{1}{2}C_{D_y}\rho\dfrac{dy}{dt}\cdot\left|\dfrac{dy}{dt}\right|S_y - mg \end{cases}$$

- これを実装する.
- ソースコードの紹介は割愛します. 1次元の時より上級者向け.

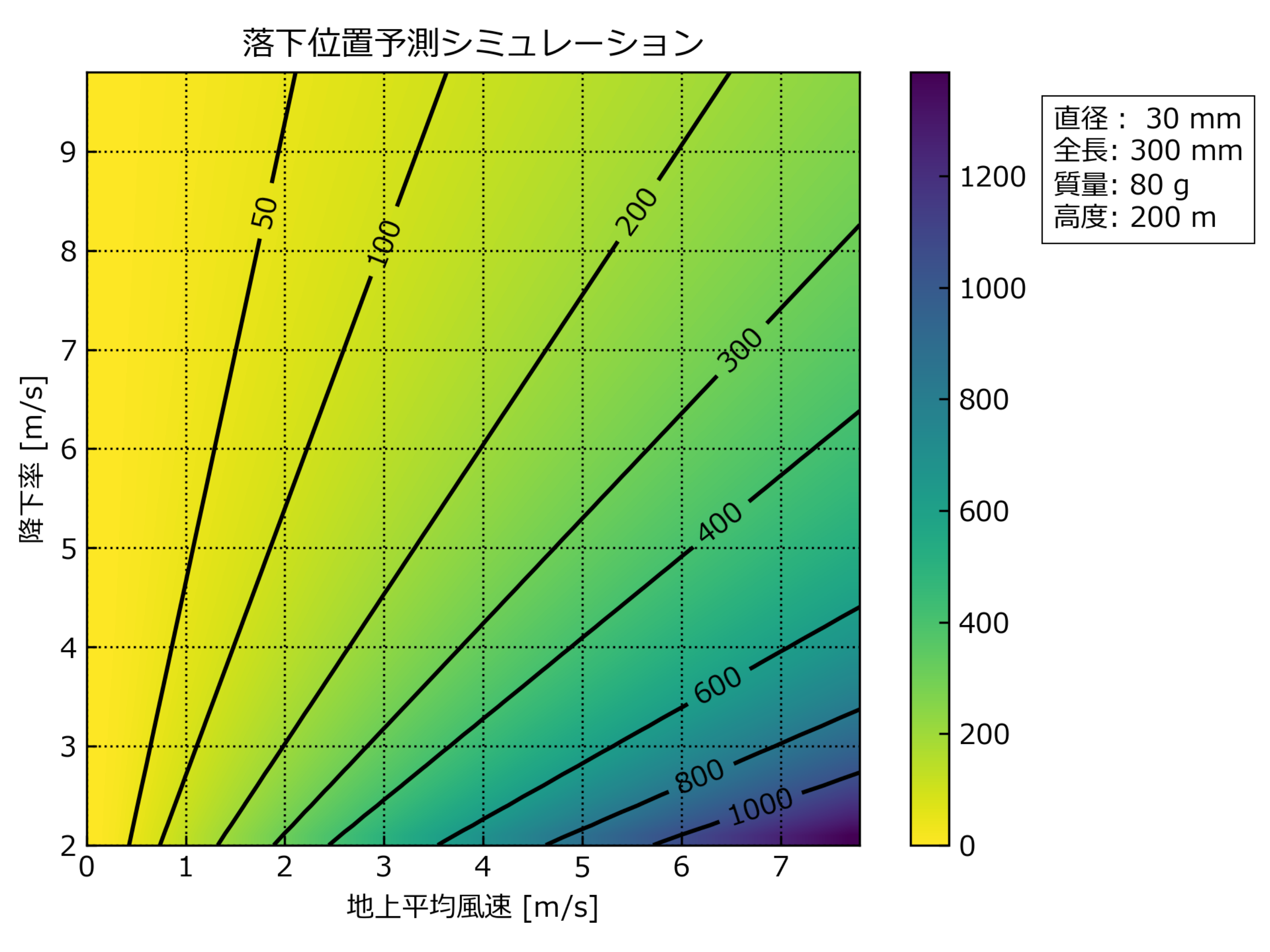
2次元2自由度系シミュレーション

- 平均風速5m/s, 射角15度で打ち上げた場合
- 風下に30m流されることがわかる.



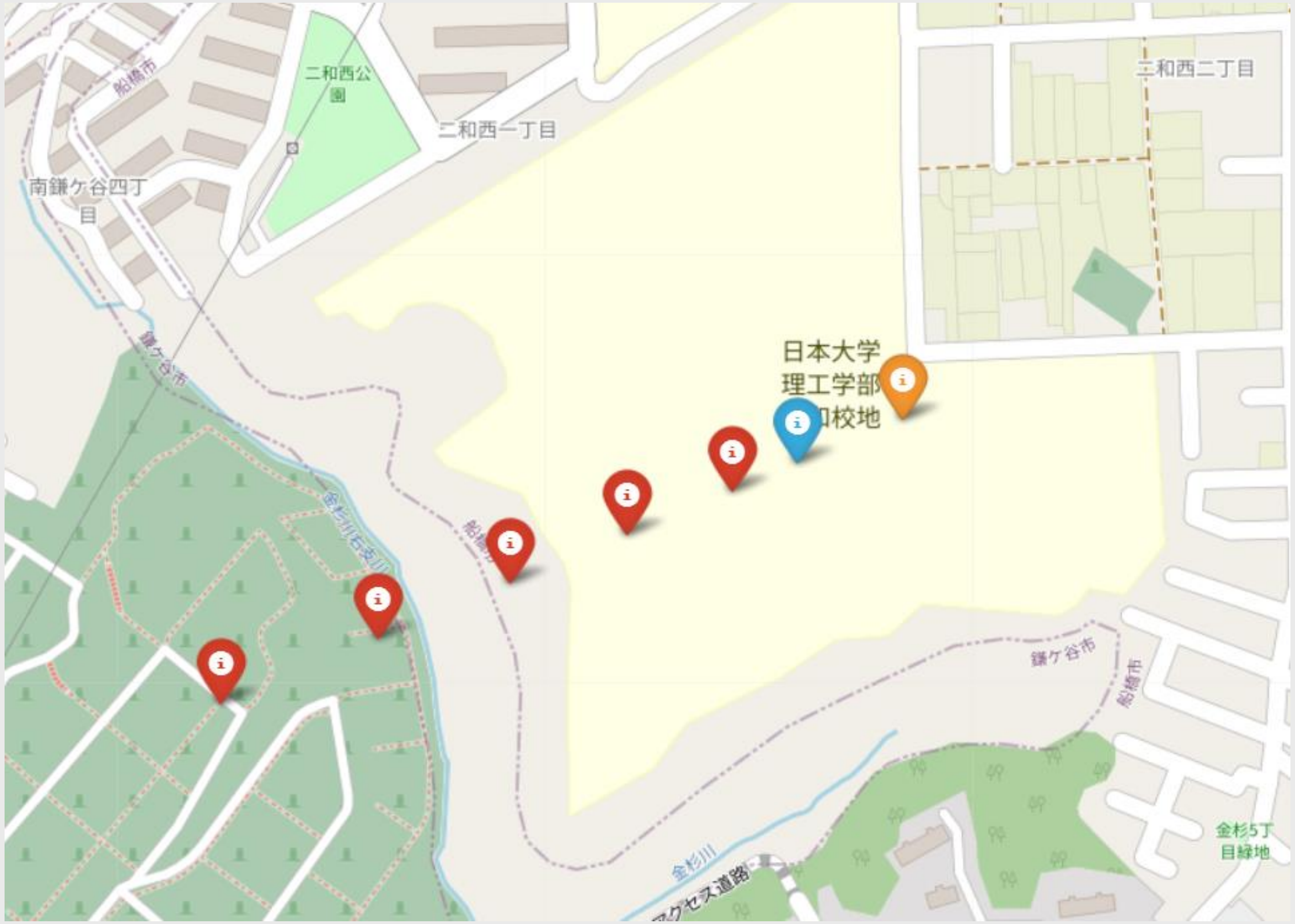
Python数値解析の応用例

- 風速/降下率が変化した場合の落下距離



Python数値解析の応用例

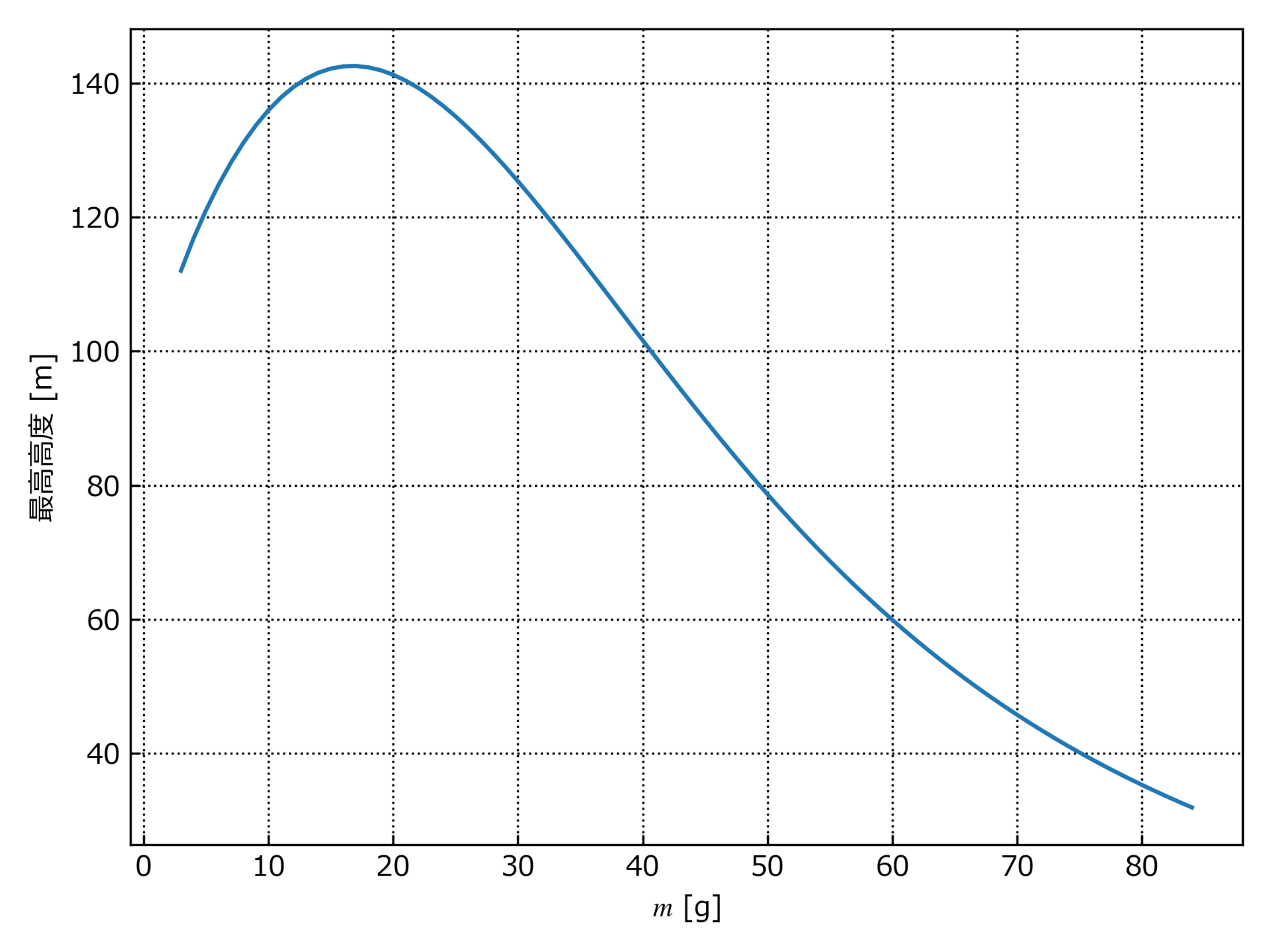
➢ 特定の風向・風速における落下位置予測



➢ 青：射点
➢ オレンジ：開傘位置
➢ 赤：1m/s, 2m/s…の落下位置

Python数値解析の応用例

➢ 機体質量ー最高高度（ A8-3, 直径25mm, $C_D = 0.75$ ）



さいごに

- モデルロケットシミュレーションの感じが少しわかったと思います.
- 1次元モデルでも, 概念設計には役に立ちます.
- ロケットに飛行を定量的に評価できます.
- 皆さん自身で色々試行錯誤してみてください.

- 講義やソースコードの間違いや改善点があったら
  ぜひNUROPの公式Xまでご連絡ください.