In [1]:
```
import numpy as np
from scipy.interpolate import interp1d
from scipy.integrate import ode
from sys import argv, exit
import argparse
```

In [2]:
```
parser = argparse.ArgumentParser(
    description='Решение задачи внешней баллистики с учетом влияния воздушных возмущений
    formatter_class=argparse.ArgumentDefaultsHelpFormatter
)
parser.add_argument('-m','--mass', type=float, default=100, help='масса груза')
parser.add_argument('-H','--height', type=float, default=1000, help='высота сброса')
parser.add_argument('-v','--velocity', type=float, default=250, help='начальная скорость
parser.add_argument('-F', type=str, default='F.csv', help='имя файла с данными об аэроди
parser.add_argument('-W', '--wind', type=str, default='Wind.csv', help='имя файла с данн
```

Out[2]:
```
_StoreAction(option_strings=['-W', '--wind'], dest='wind', nargs=None, const=None, defau
```

In [3]:
```
# TODO: при конвертации в .py заменить аргумент на sys.argv[1:]
params = parser.parse_args(input().split())
```

In [4]:
```
Fa_data = np.loadtxt(params.F, skiprows=1, delimiter=', ')
Wind_data = np.loadtxt(params.wind, skiprows=1, delimiter=', ')
```

In [5]:
```
# Гарантируем, что данные отсортированы по столбцу высоты
Fa_data = Fa_data[Fa_data[:, 0].argsort()]
Wind_data = Wind_data[Wind_data[:, 0].argsort()]
```

In [6]:
```
# Дополняем данные нулевой силой ветра по координате Y
if Wind_data.shape[1] == 3:
    Wind_data = np.insert(Wind_data, 2, values=0, axis=1)
```

In [7]:
```
# Оборачиваем данные в аппроксимирующую функцию, поддерживающую выход за границы данных
def interpolate(x_points, y_points):
    _interp = interp1d(x_points, y_points, axis=0)
    def interp(x):
        if x < x_points[0]:
            return y_points[0]
        elif x > x_points[-1]:
            return y_points[-1]
        return _interp(x)
    if y_points.ndim == 1:
        return np.vectorize(interp)
    elif y_points.ndim == 2:
        return np.vectorize(interp, signature='()->(n)')

Fa = interpolate(Fa_data[:, 0], Fa_data[:, 1])
Wind = interpolate(Wind_data[:,0], Wind_data[:, 1:])
```

In [8]:
```
def diff(time, state):
    v = state[3:]
    v_norm = np.linalg.norm(v)
    Fa_vec = v / v_norm * Fa(v_norm) if v_norm > 1e-12 else v
    wind = Wind(state[1])
    derivative = np.array([
        state[3],
        state[4],
        state[5],
        wind[0] / m - Fa_vec[0] / m,
        wind[1] / m - Fa_vec[1] / m - g,
        wind[2] / m - Fa_vec[2] / m,
    ])
    return derivative
```

In [9]:
```
def simulate(initial, dt, break_condition):
    system = ode(diff)
    system.set_initial_value(initial, 0)
    coords = [initial]
    while not break_condition(system.t, system.y):
        state = system.integrate(system.t + dt)
        coords.append(state)
    return np.array(coords)
```

In [10]:
```
# Всегда берется одинаковым образом
X, Z = 0, 0
g = 9.81

# задается пользователем.
H0 = params.height
v0 = params.velocity
m = params.mass

# может быть задано произвольно
# Поэтому выбираем такой угол, чтобы средний ветер дул по направлению к цели
alpha = np.arctan2(*Wind_data[:, [1,3]].mean(axis=0))

initial1 = [X, H0, Z, v0 * np.cos(alpha), 0, v0 * np.sin(alpha)]
```

In [11]:
```
# Берем некие начальные значения и интегрируем систему
# Финальные координаты X,Z взять с минусом как начальные условия
# В силу отсутствия зависимости от x,z должен получится эквивалентный сдвиг траектории

dt = 0.01
sim1 = simulate(initial1, dt, lambda t, state: state[1] <= 0.0)
```

In [12]:
```
# Линейной интерполяцией уточняем необходимые значения в момент Y==0
time = dt * (len(sim1)-1) + np.interp(0, sim1[:-3:-1, 1], [dt, 0.0])
```

```
        X = -np.interp(0, sim1[:-3:-1, 1], sim1[:-3:-1, 0])
        Z = -np.interp(0, sim1[:-3:-1, 1], sim1[:-3:-1, 2])
```

In [13]:
```
        # AGREEMENT
        # Проводим финальную симуляцию с найденной начальной точкой
        # и уточненными границами по времени
        initial2 = [X, H0, Z, v0 * np.cos(alpha), 0, v0 * np.sin(alpha)]
        dt = time / len(sim1)
        sim2 = simulate(initial2, dt, lambda t, s: t >= time-dt)
```

In [14]:
```
        np.savetxt(
            'Ballistic.csv',
            np.c_[np.arange(0, dt*len(sim2), dt), sim2],
            fmt='%.6f',
            header='time,X,Y,Z,Vx,Vy,Vz',
            delimiter=',',
            comments=''
        )
```