Module 5D - Crop Dusting

Omar Betancourt, Payton Goodrich, Emre Mengi

July 24, 2021

Contents

Objectives:
Develop a computational framework for aerial drops of water and pesticides in crop field environments by developing a meshless discrete element model that tracks the trajectory of released airborne materials from a controlled aircraft subjected to prevailing wind velocities. A Machine Learning Algorithm (MLA) will be wrapped around this framework to rapidly ascertain the optimal aircraft (unmanned or manned) dynamics to maximize the release effectiveness (released material usage and target impact).

Prerequisite Knowledge: N/A

Prerequisite Modules:

1A - Calculus, 1B - Linear Algebra, 1D - Differential Equations, 2C - Particle Dynamics, 3C - Generic Time Stepping, 4A - Genetic Algorithms, 4B - Gradient-based Optimization

Difficulty: Hard

Summary:

In this module, you will model and simulate aerial drops of agricultural chemicals/water on a crop field. In order to tangibly illustrate this process, this work develops a framework for a model problem combining:

1. A meshless discrete element "submodel" that tracks the trajectory of released airborne parcels ranging from powders to encapsulated water "bomblets", subjected to prevailing wind velocities.
2. A Machine Learning Algorithm (MLA) to rapidly ascertain the optimal aircraft (unmanned or manned) dynamics to maximize the material release effectiveness.

The framework is designed to enable Digital Twin type technologies, i.e. digital replicas that run in real time with the physical system. However, it is also designed to run at much faster rates, in order to enable MLA's to optimize the planning, by running quickly on laptops and mobile systems. The overall guiding motivation is to provide a useful tool for enable rapid flight-path planning for pilots *in real-time*. Numerical examples are provided to illustrate the process.

1 Theory

1.1 Model Problem

We consider the release of a random distribution of parcels into an ambient atmosphere (Figure 1.3). We assume that the parcels are small enough, relative to the scale of the overall problem, that they can be considered as particles.
Following formulations for physically similar problems associated with particulate dynamics from the fields of blasts, explosions and fire embers (Zohdi [3 ,4, 5, 1]), we make the following assumptions:

- We assume the same initial velocity magnitude for all particles under consideration, with a random distribution of outward directions away from the source. This implies that a particle non-interaction approximation is appropriate. Thus, the inter-particle collisions are negligible. This has been repeatedly verified by "brute-force" collision calculations using formulations found in Zohdi [6, 48, 7, 8].
- We assume that the particles are spherical with a random distribution of radii R_i, $i = 1, 2, 3...N = particles$. The masses are given by $m_i = \rho_i \frac{4}{3}\pi R_i^3$, where ρ_i is the density of the particles.
- We assume that the particles are *quite small* and that the amount of rotation, if any, contributes negligibly to the overall trajectory of the particles. The equation of motion for the i^{th} particle in the system is

$$m_i \dot{\boldsymbol{v}}_i = \boldsymbol{\Psi}_i^{grav} + \boldsymbol{\Psi}_i^{drag}, \tag{1.1}$$

Figure 1.1: An aerial crop duster. Courtesy of the public domain website: *https* : *//pixabay.com*

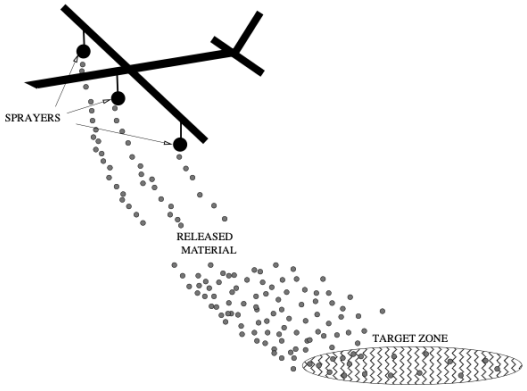

Figure 1.2: A schematic of the model problem.

with initial velocity $\boldsymbol{v}_i(0)$ and initial position $\boldsymbol{r}_i(0)$. The gravitational force is $\boldsymbol{\Psi}_i^{grav} = m_i\boldsymbol{g}$, where $\boldsymbol{g} = (g_x, g_y, g_z) = (0, 0, -9.81)\, m/s^2$.

- For the drag, we will employ a general phenomenological model

$$\boldsymbol{\Psi}_i^{drag} = \frac{1}{2}\rho_a C_D ||\boldsymbol{v}^f - \boldsymbol{v}_i||(\boldsymbol{v}^f - \boldsymbol{v}_i)A_i, \tag{1.2}$$

where C_D is the drag coefficient, A_i is the reference area, which for a sphere is $A_i = \pi R_i^2$, ρ_a is the density of the ambient fluid environment and $\boldsymbol{v}^f$ is the velocity of the surrounding medium which, in the case of interest, is air. We will assume that the velocity of the surrounding fluid medium ($\boldsymbol{v}^f$) is given, implicitly assuming that the dynamics of the surrounding medium are unaffected by the particles. [1]

In order to gain insight, initially, reader may refer back to the Particle Kinematics module on the closely related, analytically tractable, Stokesian Model.

[1] We will discuss these assumptions further, later in the work.

Remarks: As mentioned, there are a large number of physically similar phenomena in particulate dynamics associated with blasts, explosions and fire embers. We refer the interested reader to the wide array of literature on this topic; see Plimpton [9], Brock [10], Russell [11], Shimanzu [12], Werrett [13], Kazuma [14, 15], Wingerden et al [16] and Fernandez-Pello [17], Pleasance and Hart [18], Stokes [19] and Rowntree and Stokes [20], Hadden et al [21], Urban et al [22] and Zohdi [1].
Computational approaches for more complex models

1.1.1 More detailed characterization of the drag

In order to more accurately model the effects of drag, one can take into account that the empirical drag coefficient varies with Reynolds number. For example, consider the following piecewise relation (Chow [23]):

- For $0 < Re \leq 1$, $C_D = \frac{24}{Re}$,
- For $1 < Re \leq 400$, $C_D = \frac{24}{Re^{0.646}}$,
- For $400 < Re \leq 3 \times 10^5$, $C_D = 0.5$,
- For $3 \times 10^5 < Re \leq 2 \times 10^6$, $C_D = 0.000366 Re^{0.4275}$ and
- For $2 \times 10^6 < Re < \infty$, $C_D = 0.18$,

where, as in the previous section, the local Reynolds number for a particle is $Re \stackrel{\text{def}}{=} \frac{2R_i \rho_a ||\boldsymbol{v}^f - \boldsymbol{v}_i||}{\mu_f}$ and μ_f is the fluid viscosity.[2] *We note that in the zero Reynolds number limit, the drag is Stokesian.* In order to solve the governing equation,

$$m_i \dot{\boldsymbol{v}}_i = \boldsymbol{\Psi}_i^{grav} + \boldsymbol{\Psi}_i^{drag} = m_i \boldsymbol{g} + \frac{1}{2} \rho_a C_D ||\boldsymbol{v}^f - \boldsymbol{v}_i|| (\boldsymbol{v}^f - \boldsymbol{v}_i) A_i, \tag{1.3}$$

we integrate the velocity numerically

$$\boldsymbol{v}_i(t + \Delta t) = \boldsymbol{v}_i(t) + \frac{1}{m_i} \int_t^{t+\Delta t} (\boldsymbol{\Psi}_i^{grav} + \boldsymbol{\Psi}_i^{drag}) \, dt \approx \boldsymbol{v}_i(t) + \frac{\Delta t}{m_i} \left(\boldsymbol{\Psi}_i^{grav}(t) + \boldsymbol{\Psi}_i^{drag}(t) \right). \tag{1.4}$$

The position is the obtained by integrating again:

$$\boldsymbol{r}_i(t + \Delta t) = \boldsymbol{r}_i(t) + \int_t^{t+\Delta t} \boldsymbol{v}_i(t) \, dt \approx \boldsymbol{r}_i(t) + \Delta t \boldsymbol{v}_i(t). \tag{1.5}$$

This approach has been used repeatedly for a variety of physically similar drift-type problems in Zohdi [3, 4, 5, 1].
Remark 1: The piecewise drag law of Chow [23] is a mathematical description for the Reynolds number over a wide range and is a curve-fit of extensive data from Schlichting [24].
Remark 2: Numerical simulations using this algorithm for scenarios involving explosions can be found in Zohdi [5] and Zohdi and Cabalo [4]. In Zohdi and Cabalo [4], a systematic study of an explosive device was considered combining the results from a set of full scale field experiments with high explosives and ballistic gelatin and reduced order fragment tracking models, similar to those presented in this section, neglecting the interaction between the shock wave and the packed fragments and any chemical aspects. [3] There were extremely close matches between experiments and numerics, indicating that it is likely that a drag-based particle noninteraction model is appropriate.[4]

[2] The viscosity coefficient for air is $\mu_f = 0.000018$ Pa-s.
[3] For shock analyses, see, for example, Hoover and Hoover [25], Gregoire et al [26], Kudryashova et al [27], Cabalo et al [28, 29].
[4] One conclusion from these experiments was that aerosols generated from a blast containing toxic materials cannot be assumed to be inactivated by the blast itself, which is consistent with findings of Eshkol and Katz [??], and Kanemitsu [??], where Hepatitis B from a suicide bomber was transmitted to survivors of the blast.

Remark 3: We emphasize that there are different ways for hot particles to be generated. One way is by power line (generally aluminum or copper) interactions in high-winds (referred to as buffeting or galloping) which arc or clash (Pleasance and Hart [18], Russell et al [30] and Blackburn [31]). In such situations, metal fragments may be produced and ejected from the arcing location (Pagni [32], Gilbert [33], Maraghides and Mell [34] and Ramljak et al. [35] and Pleasance and Hart [18]).

1.1.2 Modeling aircraft dynamics

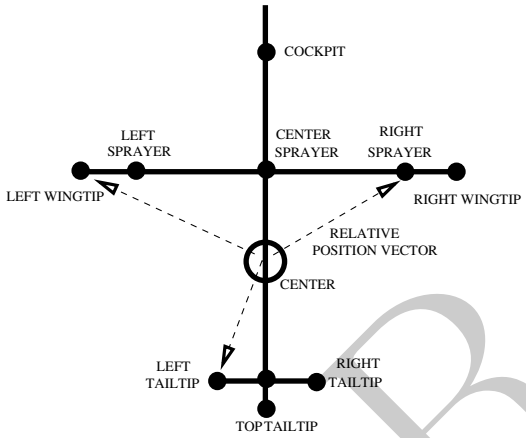

Figure 1.3: The fuselage layout and position vectors to various locations. The entire body is governed by rigid body kinematics with two variables: the angular velocity of the body $\boldsymbol{\omega}$ and the velocity of the center $\boldsymbol{v}_c$. The velocities and positions of all other points can be determined by rigid body kinematics and integration.

1.1.3 Fuselage dynamics

We assume two control variables: (1) The angular velocity of the aircraft and (2) its speed. The relative velocity of any point p on the aircraft with respect to the center c is (Figure 1.3)

$$\boldsymbol{v}_{c\to p} = \boldsymbol{\omega} \times \boldsymbol{r}_{c\to p}. \tag{1.6}$$

The total velocity is

$$\boldsymbol{v}_p = \boldsymbol{v}_c + \boldsymbol{v}_{c\to p}. \tag{1.7}$$

Consequently, the new position is related to the velocity by

$$\frac{d\boldsymbol{r}_p}{dt} = \boldsymbol{v}_p \tag{1.8}$$

and thus

$$\boldsymbol{r}_p(t+\Delta t) = \boldsymbol{r}_p(t) + \int_t^{t+\Delta t} \boldsymbol{v}_p\, dt \approx \boldsymbol{r}_p(t) + \boldsymbol{v}_p(t)\Delta t \tag{1.9}$$

For this equation, we can ascertain any point's position relative to that of the center.

1.1.4 Flow rates for parcel release

The total number of parcels dropped is given by

$$N_p = \zeta(T_f - T_i) \tag{1.10}$$

The radius of a parcel is a (volume) control variable, R_p. The mass of the parcel is given by

$$m_p = \rho_p \frac{4}{3} \pi R_p^3. \tag{1.11}$$

2 Example

2.1 A numerical example

In order to illustrate the model, the following simulation parameters were chosen:

- Starting height of 200 meters,
- Total simulation duration, 40 seconds,
- The time step size, $\Delta t = 10^{-6}$ seconds,
- The release velocity, $\boldsymbol{v}(t=0) = 30\, m/s$,
- Density of material being released , $\rho_i = 1000$, kg/m^3,
- Density of air, $\rho_a = 1.225$, kg/m^3 and
- Total mass, $M^{Total} = \sum_{i=1}^{P_n} m_i = \ kg$.

Figures 2.1-**??** illustrate the action of the the crop duster and the resulting hits on the target. Approximately 19 % of the targets were hit. A "hit" was computed as follows:

$$||\boldsymbol{r}_i - \boldsymbol{T}_j|| \leq \mathcal{F}(R_i^r, R_j^t), \tag{2.1}$$

where

$$\mathcal{F}(R_i^t, R_j^r) = \beta R_i^t + R_i, \tag{2.2}$$

where β_i represents the "splatter" of the water (for example $\beta = 10$). 1000 targets were used.

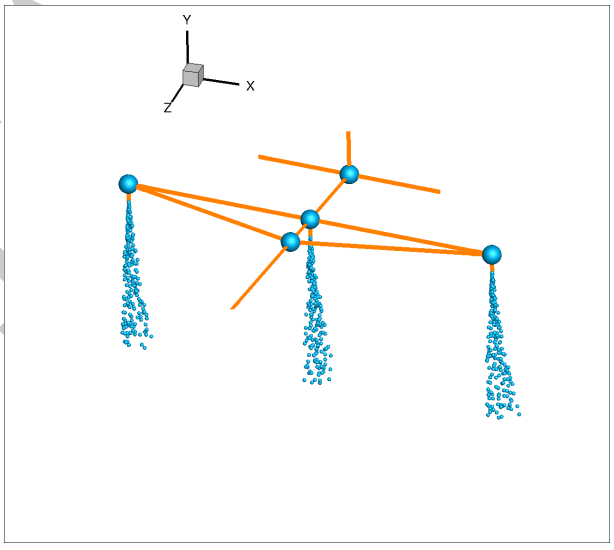

Figure 2.1: Zoom on the aerial simulation model problem aircraft.

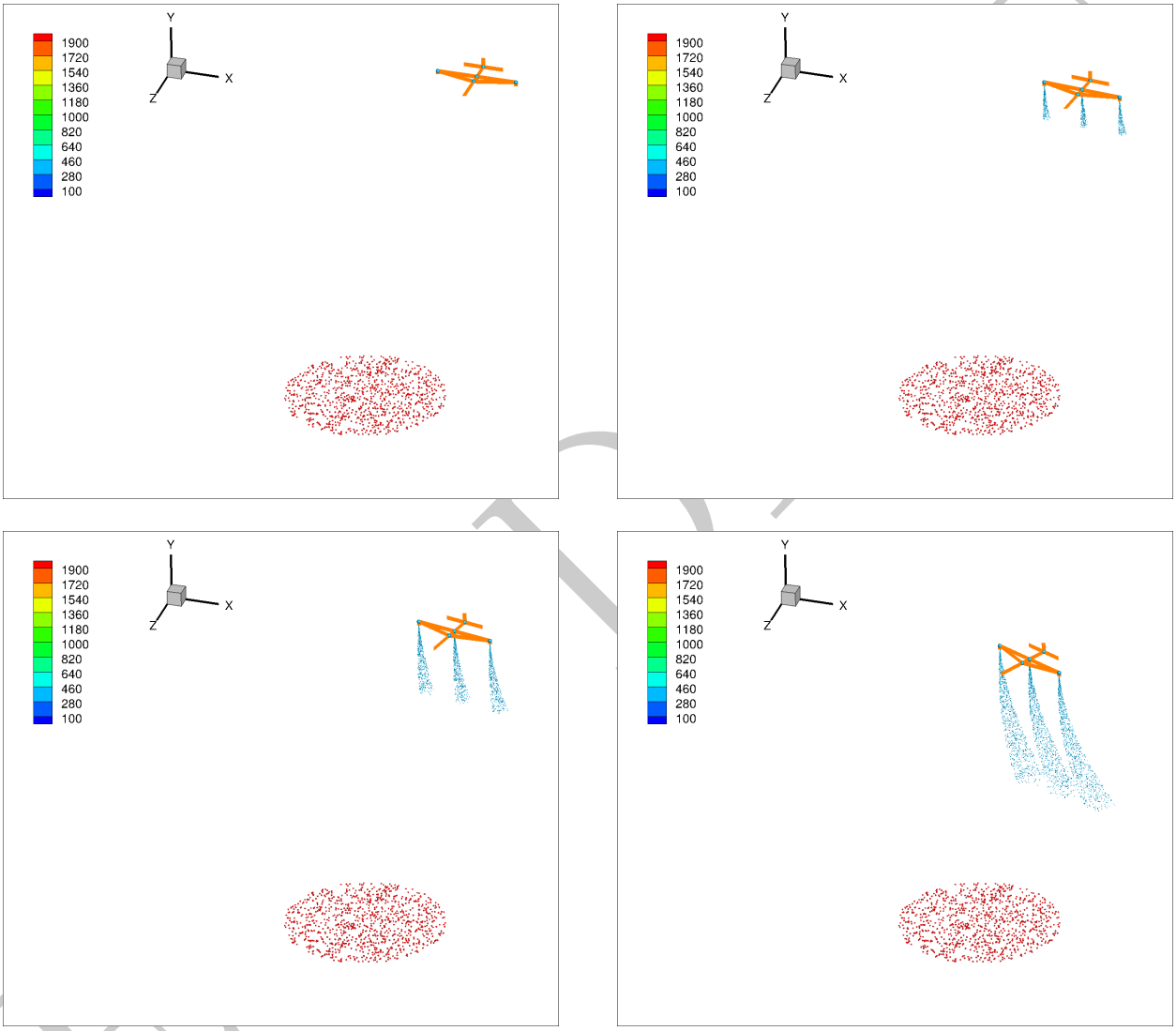

Figure 2.2: Aerial simulation frames with 1000 targets. Most of the release material did not land on the target patch. This motivates the next section of Machine-Learning to seek more effective strategies.

2.2 Machine-Learning for system optimization

The speed at which this type of simulation can be completed allows one to answer the inverse question of what the combination of parameters should be for a desired result. In order to cast the objective mathematically, we set the problem up as a Machine Learning Algorithm (MLA); specifically a Genetic Algorithm (GA) variant. Following Zohdi [43, 44, 49, 1] , we formulate the objective as a cost function minimization problem whereby simulation parameters are adjusted to match desired responses as closely as possible, given by the following cost function

$$\Pi^{tot}(\lambda_1, ...\lambda_{13}) = w_1\Pi^{(1)} + w_2\Pi^{(2)} \tag{2.3}$$

where

$$\Pi^{(1)} = (1 - \frac{T^h}{T^{tot}}) \qquad \text{and} \qquad \Pi^{(2)} = (1 - \frac{P^h}{P^{tot}}), \tag{2.4}$$

where T^{tot} is the total number of targets and T^h is the total number of targets hit and where P^{tot} is the total number of parcels released and P^h is the total number of parcels that hit the targets. We systematically minimize Equation 2.3, $min_\Lambda \Pi$, by varying the design parameters: $\mathbf{\Lambda}^i \stackrel{\text{def}}{=} \{\Lambda_1^i, \Lambda_2^i, \Lambda_3^i, ..., \Lambda_N^i\} \stackrel{\text{def}}{=} \{flow\, rate, aircraft\, dynamics...\}$. The system parameter search is conducted within the constrained ranges of $\Lambda_1^{(-)} \leq \Lambda_1 \leq \Lambda_1^{(+)}$, $\Lambda_2^{(-)} \leq \Lambda_2 \leq \Lambda_2^{(+)}$ and $\Lambda_3^{(-)} \leq \Lambda_3 \leq \Lambda_3^{(+)}$, etc. These upper and lower limits would, in general, be dictated by what is physically feasible.
For specifics in setting up a GA for this framework, please refer to the Genetic Algrothim Module.

3 Assignment

For this assignment, create a simulation of a crop duster and apply the MLA algorithm, with the following design objectives
Set up the cost function as:

$$\Pi^{tot} = w_1\Pi^{(1)} + w_2\Pi^{(2)} \tag{3.1}$$

where

$$\Pi^{(1)} = (1 - \frac{T^h}{T^{tot}}) \tag{3.2}$$

where T^{tot} is the total number of targets and T^h is the total number of targets hit and

$$\Pi^{(2)} = (1 - \frac{P^h}{P^{tot}}) \tag{3.3}$$

where P^{tot} is the total number of parcels released and P^h is the total number of parcels that hit the targets.
Set up the Machine-Learning parameters (total of thirteen) to be:

1. Λ_1 =the plane's velocity (magnitude): v^{plane},
2. $\Lambda_2, \Lambda_3, \Lambda_4$ = the plane's angular velocity: $^{plane} = (\omega_x, \omega_y, \omega_z)$,
3. $\Lambda_5, \Lambda_6, \Lambda_7$ =the initial plane position, $(0) = (r_x(0), r_y(0), r_z(0))$,
4. Λ_8 = the particle size, R_i,
5. Λ_9 = the sprayer amplitude A,
6. Λ_{10} = the release time start, T_I,
7. Λ_{11} = the release time end, T_F,
8. Λ_{12} = the drop rate of retardant: ζ and
9. Λ_{13} = the particle drop velocity, v^{drop}.

Explicitly, the design vector is:

$$\Lambda = \{v^{plane}, \omega_x, \omega_y, \omega_z, r_x(0), r_y(0), r_z(0), R_i, A, T_i, T_e, \zeta, v^{drop}\}. \tag{3.4}$$

The initial search range:

1. $100km/hr = 100 \times 1000/3600m/s = \Lambda_1^- \leq \Lambda_1^- \leq \Lambda_1^+ = 1000km/hr = 1000 \times 1000/3600m/s$,
2. $(-2,-2,-2)rad/s = (\Lambda_2^-, \Lambda_3^-, \Lambda_4^-) \leq (\Lambda_2, \Lambda_3, \Lambda_4) \leq (\Lambda_2^+, \Lambda_3^+, \Lambda_4^+) = (2,2,2)rad/s$,
3. $(-100, 100, -100)m = (\Lambda_5^-, \Lambda_6^-, \Lambda_7^-) \leq (\Lambda_5, \Lambda_6, \Lambda_7) \leq (\Lambda_5^+, \Lambda_6^+, \Lambda_7^+) = (100, 250, 100)m$,
4. $0.05m\Lambda_8^- \leq \Lambda_8 \leq \Lambda_8^+ = 0.2m$,
5. $0.0 = \Lambda_9^- \leq \Lambda_9 \leq \Lambda_9^+ = 0.5$,
6. $0.0T = \Lambda_{10}^- \leq \Lambda_{10} \leq \Lambda_{10}^+ = 0.25T$,
7. $0.25T = \Lambda_{11}^- \leq \Lambda_{11} \leq \Lambda_{11}^+ = T$,
8. $10m^3/s = \Lambda_{12}^- \leq \Lambda_{12} \leq \Lambda_{12}^+ = 60m^3/2$,
9. $100m/s = \Lambda_{13}^- \leq \Lambda_{13} \leq \Lambda_{13}^+ = 400m/s$,

Also use

- Time: $T = 2.5$ seconds and
- Design weights: w_1 and $w_2 = 1$.

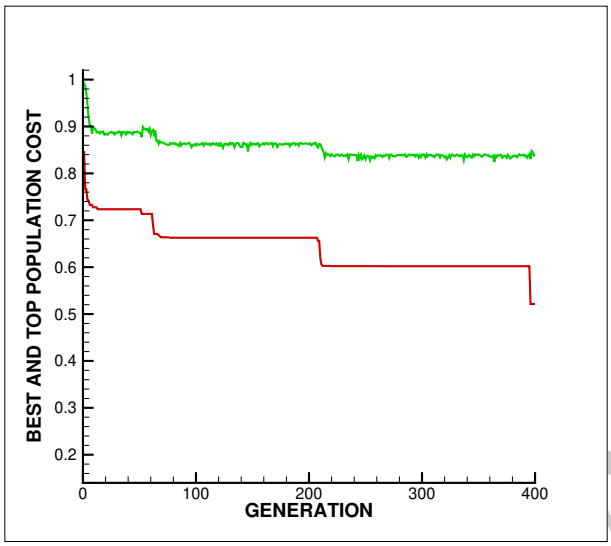

Figure 3.1: The reduction of the cost function for the 13 parameter set. Shown are the best performing gene (design parameter set, in *red*) as a function of successive generations, as well as the average performance of the entire population of genes (designs, in *green*)

Figure 3.1 shows the reduction of the cost function for the 13 parameter set. This cost function Π represents the percentage of unused material left in the zone of interest. In other words, the system is being driven to the parameters generating the worst case scenario. Shown are the best performing gene (design parameter set, in *red*) as a function of successive generations, as well as the average performance of the entire population of the genes (designs, in *green*). Use the following MLA settings:

- Number of design variables: 13,
- Population size per generation: 50,
- Number of parents to keep in each generation: 10,
- Number of children created in each generation: 10,
- Number of completely new genes created in each generation: 30
- Number of generations for re-adaptation around a new search interval: 20 and
- Number of generations: 400.

4 Solution Code

```

clc;
clear;
close all;

% v = VideoWriter('bomberVideo', 'MPEG-4');
% open(v);

%%%%%%%%%%%%%%%%%%%%%%%%%% GENETIC ALGORITHM %%%%%%%%%%%%%%%%%%%%%%%%%%
G = 400;  % Total Number of Generations
S = 50;   % Number of strings per generation


Pi = zeros(S,1);

nParents = 10;
nChildren = 10;

Lm(1) = 100*1000/3600;
Lp(1) = 1000*1000/3600;
Lm(2) = -2; Lm(3) = -2; Lm(4) = -2;
Lp(2) = 2; Lp(3) = 2; Lp(4) = 2;
Lm(5) = -100; Lm(6) = 100; Lm(7) = -100;
Lp(5) = 100; Lp(6) = 250; Lp(7) = 100;
Lm(8) = 0.05;
Lp(8) = 0.2;
Lm(9) = 0.0;
Lp(9) = 0.5;
Lm(10) = 0.0;
Lp(10) = 0.25*2.5;
Lm(11) = 0.25*2.5;
Lp(11) = 2.5;
Lm(12) = 10;
Lp(12) = 60;
Lm(13) = 100;
Lp(13) = 400;

Lam = Lm(:) + (Lp(:) - Lm(:)).*rand(13,S); % 13 params by 20 rand numbers from 0 to 1


for g = 1:G
    g
    for s = 1:S
        s
        [Pi(s)] = myCROPDUSTER(Lam(:,s));
    end

    % SORT
    [Pi, ind] = sort(Pi);
    Lam = Lam(:,ind);

    % MATE
    phi = rand(13,nChildren);
    ind1 = 1:2:nChildren;
    ind2 = 2:2:nChildren;

    topParents = Lam(:,1:nParents); % keep top k parents
    kidOneFromTopParents = phi(ind1).*Lam(:,ind1) + (1 - phi(ind1)).*Lam(:,ind2);
    kidTwoFromTopParents = phi(ind2).*Lam(:,ind2) + (1 - phi(ind2)).*Lam(:,ind1);
    newKids = Lm + (Lp - Lm)*rand(13,S-2*nParents);
    Lam = horzcat(topParents, kidOneFromTopParents, kidTwoFromTopParents, newKids);

    Pi(1)
end

```

```
function [PI] = myCROPDUSTER(Params)
numPlanes = 1;
numTargets = 1000;
plane_params = containers.Map({'vPlane',   'wy',     'wz',     'wx',    'ry',      'rz',
    'rx',  'sprayAmp'},...
                              {Params(1),Params(2),Params(3),Params(4),Params(5),Params(6),
    Params(7), Params(9)});

sprayer_params = containers.Map({'size',  'sprayAmp',   'Ti',     'Tf',     'dropRate', '
    dropVel'},...
                                {Params(8),Params(9),Params(10),Params(11),Params(12),Params
    (13)});

Ti = sprayer_params('Ti');          % Starting spray time
Tf = sprayer_params('Tf');          % Ending spray time
volumeDropRate = sprayer_params('dropRate'); % Drop rate
radParticle = sprayer_params('size'); % Radius of each particle

totalVolumeParticles = volumeDropRate*(Tf-Ti);
particleVolume = 4/3*pi*radParticle^3;
totalNumParticles = floor(totalVolumeParticles/particleVolume);
dtParcels = (Tf-Ti)/totalNumParticles; % Delta time between parcels

nTimeSteps = floor(2.5/dtParcels);

% Initializing collection of planes
planes(numPlanes) = {NaN};

% Creating plane object
for i = 1:numPlanes
    planes(i) = {Plane(plane_params, dtParcels)};
end

% % Init. 3D Fig.
% fig1 = figure('pos', [0 200 1000 1000]);
% h = gca;
% view(135, 30);
%
% axis equal;
% grid on;
%
% xlim([0 250]);
% ylim([0 250]);
% zlim([0 200]);
% xlabel('X[m]');
% ylabel('Y[m]');
% zlabel('Z[m]');
% hold(gca, 'on');

% Allocating memory for all particles' pos and vel
posParticle = NaN(3,totalNumParticles);
velParticle = NaN(3,totalNumParticles);

% Generating targets
R = 65; % Radius of map area taken by targets
targetCenter = [125,125];
t = 2*pi*rand(1,numTargets);
r = R*sqrt(rand(1,numTargets));

% Target positions
xPosTarget = targetCenter(1) + r.*cos(t);
yPosTarget = targetCenter(2) + r.*sin(t);
zPosTarget = zeros(1,numTargets);
posTarget = [xPosTarget; yPosTarget; zPosTarget];

for i = 1:numPlanes
    drone_body = planes{i}.GetBody();
%     fig_plane(i) = scatter3(gca, drone_body(1,:), drone_body(2,:), drone_body(3,:), '
```

```
    MarkerEdgeColor','k', 'MarkerFaceColor',[0.8500 0.3250 0.0980]);
%     fig_shadow(i) = plot3(gca, 0, 0, 0, 'xk', 'LineWidth', 3);
%     fig_particles(i) = scatter3(gca, posParticle(1,:), posParticle(2,:), posParticle(3,:),
     'MarkerEdgeColor','k', 'MarkerFaceColor',[0.3010 0.7450 0.9330]);
%     fig_targets(i) = scatter3(gca, posTarget(1,:), posTarget(2,:), posTarget(3,:), '
    MarkerEdgeColor','k', 'MarkerFaceColor',[0.4660 0.6740 0.1880]);
end

mi = 1000*particleVolume;
g = 9.81;
Fg = [0; 0; -mi*g];
vf = [0; 0; 0]; %velocity of fluid (air)
rhoa = 1.225; % Surrounding Medium Density (kg/m^3)
muf = 1.8E-5; % Surrounding Medium viscosity (Pa/s)
Air = true(1,size(posParticle,2));
Ai = pi*radParticle^2;  % Drag Reference Area
vDrop = sprayer_params('dropVel');  % Initial drop Velocity

count = 0;
% Event Loop

for k = 1:nTimeSteps
    % Looping through all planes
    for i = 1:length(planes)
        planes{i}.UpdateState();
        if ((Ti <= k*dtParcels) && (k*dtParcels <= Tf))
            count = count + 1;
            planes{i}.Sprayer();
            sprayerState = planes{i}.GetSprayer();
            sprayerPos = sprayerState(:,1);
            sprayerVel = sprayerState(:,2);
            sprayerNorm = sprayerState(:,3);

            velParticle(:,count) = sprayerVel + vDrop*sprayerNorm;
            posParticle(:,count) = sprayerPos + dtParcels*velParticle(:,count);
        end
    end

    if Ti <= k*dtParcels
        if any(Air)
            vdiff = vecnorm(vf-velParticle(:,Air),2,1);
            Re = (2*radParticle*rhoa*vdiff)./muf;
            Cd = (24./Re).*(0 < Re & Re <= 1) + ...
                (24./(Re.^(0.0646))).*(1 < Re & Re <= 400) + ...
                0.5.*(400 < Re & Re <= 3E5) + ...
                (0.000366*(Re.^(0.4275))).*(3E5 < Re & Re <= 2E6) + ...
                0.18.*(2E6 < Re);

            Fd = 0.5.*Cd.*rhoa.*Ai.*vdiff.*(vf-velParticle(:,Air));

            Ftot = Fd + Fg;

            posParticle(:,Air) = posParticle(:,Air) + velParticle(:,Air)*dtParcels; % Use
    Forward Euler to update position
            velParticle(:,Air) = velParticle(:,Air) + dtParcels*(Ftot/mi); % Use Forward
    Euler to update velocity
            below = find(posParticle(3,:) < 0); % Check if droplets have hit the print bed
            posParticle(3,below) = 0; % Reset droplets height to print bed
            Air(below) = false; % Update flag array to indicate droplets on bed
        end
    end

%         if mod(k,10) == 0 % update movie frame
%             figure(1)
%             bodyParts = planes{i}.GetBody();
%             set(fig_plane(i), ...
%             'xData', bodyParts(1,:), ...
%             'yData', bodyParts(2,:), ...
```

```
%              'zData', bodyParts(3,:));
%              set(fig_shadow(i), ...
%              'xData', bodyParts(1), ...
%              'yData', bodyParts(2), ...
%              'zData', 0);
%              set(fig_particles(i), ...
%              'xData', posParticle(1,:), ...
%              'yData', posParticle(2,:), ...
%              'zData', posParticle(3,:));
% %              frame = getframe(gcf);
% %              writeVideo(v,frame);
%          end
end

% Post Processing
% Calculting Inter-point distances
beta = 40; % splatter
radTarget = 1;
distCollision = beta*radParticle + radTarget;
IdxTar = rangesearch(posTarget(1:2,:)',posParticle(1:2,:)',distCollision,'SortIndices',false);
IdxPar = rangesearch(posParticle(1:2,:)',posTarget(1:2,:)',distCollision,'SortIndices',false);
posTarget(:,[IdxTar{:,1}]) = [];
posParticle(:,[IdxPar{:,1}]) = [];

% Calculate cost:
Pi1 = (1 - (numTargets - size(posTarget,2))/numTargets);
Pi2 = (1 - (totalNumParticles - size(posParticle,2))/totalNumParticles);
PI = Pi1 + Pi2;

% figure(1)
% set(fig_targets(i), ...
% 'xData', posTarget(1,:), ...
% 'yData', posTarget(2,:), ...
% 'zData', posTarget(3,:));
% frame = getframe(gcf);
% writeVideo(v,frame);

end
% close(v);


%%%%%%%%%%%%%%%% Plane.m %%%%%%%%%%%%%%%%%%%
classdef Plane < handle
    % MEMBERS
    properties % Used for dynamics
        dt;   % time step size   [s]
        x;    % state vector     [X-pos, Y-pos, Z-pos, dX, dY, dZ, p, q, r]
        r;    % position vector  [X-pos, Y-pos, Z-pos]
        dr;   % velocity vector  [dX, dY, dZ]
        w;    % drone ang. vel.  [p, q, r]
        wMat;
        body;
        sprayAmp;
        nSpray;
        velSprayer;
        posSprayer;
    end

    methods
        function obj = Plane(params, deltaTime)
            % Initializing state vector
            obj.r   = [params('rx'), params('ry'), params('rz')]';   % Plane's initial
    center position
            obj.dr  = [0, params('vPlane'), 0]';   % Plane's velocity
            obj.w   = [params('wx'), params('wy'), params('wz')]';   % Plane's angular
    Velocity
```

```
257             obj.sprayAmp = params('sprayAmp'); % Sprayer's Amplitude
258
259             % Assembling the plane
260             obj.body = [obj.r(1)+0.0,   obj.r(2)+15.0,   obj.r(3)+0.0; ...  % Cockpit
261                         obj.r(1)-20.0,  obj.r(2)+5.0,    obj.r(3)+0.0; ...  % Left Wingtip
262                         obj.r(1)+20.0,  obj.r(2)+5.0,    obj.r(3)+0.0; ...  % Right Wingtip
263                         obj.r(1)+0.0,   obj.r(2)-15.0,   obj.r(3)+0.0; ...  % Center Tail
264                         obj.r(1)-5.0,   obj.r(2)-15.0,   obj.r(3)+0.0; ...  % Left Tailtip
265                         obj.r(1)+5.0,   obj.r(2)-15.0,   obj.r(3)+0.0; ...  % Right Tailtip
266                         obj.r(1)+0.0,   obj.r(2)-15.0,   obj.r(3)+5.0; ...  % Top Tailtip
267                         obj.r(1)-20.0,  obj.r(2)+5.0,    obj.r(3)+0.0; ...  % Left Sprayer
268                         obj.r(1)-20.0,  obj.r(2)+5.0,    obj.r(3)-2.0; ...  % Left Sprayer
    End
269                         obj.r(1)+0.0,   obj.r(2)+5.0,    obj.r(3)+0.0; ...  % Center Sprayer
270                         obj.r(1)+0.0,   obj.r(2)+5.0,    obj.r(3)-2.0; ...  % Center Sprayer
     End
271                         obj.r(1)+20.0,  obj.r(2)+5.0,    obj.r(3)+0.0; ...  % Right Sprayer
272                         obj.r(1)+20.0,  obj.r(2)+5.0,    obj.r(3)-2.0]';    % Right Sprayer
    End
273
274             obj.posSprayer = [obj.r(1)+0.0,   obj.r(2)+5.0,    obj.r(3)-2.0]';
275
276             % Resizing omega into a matrix
277             obj.wMat = repmat(obj.w,1,size(obj.body,2));
278
279             obj.dt = deltaTime;
280         end
281
282         function BodyState = GetBody(obj)  % Getter function for position of plane parts
283             BodyState = obj.body;
284         end
285
286         function obj = UpdateState(obj)
287             wXr = cross(obj.wMat, obj.body);
288           obj.body = obj.body + (obj.dr + wXr)*obj.dt;
289
290             % Velocity of center sprayer
291             obj.velSprayer = obj.dr + cross(obj.w, obj.body(:,11));
292             % Position of center sprayer
293             obj.posSprayer = obj.posSprayer + obj.velSprayer*obj.dt;
294         end
295
296         function obj = Sprayer(obj)
297             centerSprayVec = obj.body(:,11) - obj.body(:,10);   % End of sprayer - Start of
    sprayer
298             centerSprayNorm = centerSprayVec./vecnorm(centerSprayVec);
299             rand3vec = [rand, rand, rand]';
300             nRand = rand3vec/norm(rand3vec); % random normal vector
301             obj.nSpray = (centerSprayNorm + obj.sprayAmp*nRand)/norm(centerSprayNorm + obj.
    sprayAmp*nRand);
302         end
303
304         function SprayerState = GetSprayer(obj)  % Getter function for sprayer position,
    velocity, and direction
305             SprayerState = [obj.posSprayer, obj.velSprayer, obj.nSpray];
306         end
307
308     end
309 end
```

5 References

1. Zohdi, T. I., (2020) A machine-learning framework for rapid adaptive digital-twin based fire-propagation simulation in complex environments. Computer Methods in Applied Mechanics and Engineering. https://doi.org/10.1016/j.cma.2020.112907

2. Schulthess, P., Neuenschwander, M, Mosalam, K. M. and Knobloch, M. (2020) A computationally rigorous approach to hybrid fire testing. Computers and Structures. Volume 238, 1 October 2020, 106301.

3. Zohdi, T. I. (2016). A note on firework blasts and qualitative parameter dependency. Proceedings of the Royal Society.DOI: 10.1098/rspa.2015.0720

4. Zohdi, T. I. and Cabalo, J. (2017). On the thermomechanics and footprint of fragmenting blasts. International Journal of Engineering Science. 118, 28-39.

5. Zohdi, T. I. (2018). Modeling the spatio-thermal fire hazard distribution of incandescent material ejecta in manufacturing. Computational Mechanics. https://doi.org/10.1007/s00466-018-1617-2

6. Zohdi, T. I. (2007). Computation of strongly coupled multifield interaction in particle-fluid systems. *Computer Methods in Applied Mechanics and Engineering.* Volume 196, 3927-3950.

7. Zohdi, T. I. (2013) Numerical simulation of charged particulate cluster-droplet impact on electrified surfaces. *Journal of Computational Physics.* 233, 509-526.

8. Zohdi, T. I. (2014) Embedded electromagnetically sensitive particle motion in functionalized fluids. Computational Particle Mechanics. 1: 27-45.

9. Plimpton, G. (1984). Fireworks: A History and Celebration. Doubleday. ISBN 0-385-15414-3.

10. Brock, A. St. Hill (1949). A History of Fireworks. George G. Harrap and Co.

11. Russell, M. S (2008). The chemistry of fireworks. Royal Society of Chemistry, Great Britain. ISBN 978-0-85404-127-5.

12. Shimizu, T. (1996). Fireworks: The Art, Science, and Technique. Pyrotechnica Publications. ISBN 978-0-929388-05-2.

13. Werrett, S. (2010). Fireworks: Pyrotechnic Arts and Sciences in European History. University of Chicago Press. ISBN 978-0-226-89377-8.

14. Kazuma, S. (2004). Hanabi no Hon. Fireworks Book . Tankosha. ISBN 4-473-03177-2.

15. Kazuma, S. (2011). Wonder of Fireworks. Soft Bank Creative. ISBN 978-4-7973-6450-7.

16. Wingerden, V. K.,Hesby, I. and Eckhoff, R. (2011), Ignition of Dust Layers by Mechanical Sparks, in: Proceedings of 7th Global Congress on Process Safety, Chicago,Ill, 2011.

17. Fernandez-Pello, A. C. (2017). Wildland fire spot ignition by sparks and firebrands. Fire Safety Journal, 91 2-10.

18. Pleasance, G. E. and Hart, J. A (1977), An Examination of Particles from Conductors Clashing as Possible Source of Bushfire Ignition. State Electricity Commission of Victoria (SEC), Victoria, Australia, Research and Development Department, Report FM-1, 1977.

19. Stokes, A. D., Fire ignition by copper particles of controlled size (1990) J. Electr. Electron. Eng., Aust. 10, 188-194.

20. Rowntree, G. and Stokes, A. (1994), Fire ignition of aluminum particles of controlled size, J. Electr. Electron. Eng. (1994) 117-123.

21. Hadden, R., Scott, S., Lautenberger, C. and Fernandez-Pello, C. A.(2011). Ignition of combustible fuel beds by hot particles: an experimental and theoretical study, Fire Technol. 47 341-355.

22. Urban, J. L., Zak, C. D., Song, J. and Fernandez-Pello, A. C., (2017). Smolder spot ignition of natural fuels by a hot metal particle, Proc. Combust. Inst. 36 (2) 3211-3218 ISSN 1540-7489 *https* : *//doi.org*/10.1016/*j.proci*.2016.09.014.

23. Chow, C. Y. (1980). An introduction to computational fluid dynamics. New York, Wiley.

24. Schlichtling, H. (1979). Boundary-layer theory. 7th edition. New York: McGraw-Hill.

25. Hoover, W. G. and Hoover, C. G. (2009). Tensor temperature and shock-wave stability in a strong two-dimensional shock wave, Phys. Rev. E: Stat., Nonlinear, Soft Matter Phys. 80, 011128/1-011128/6.

26. Gregoire, Y., Sturtzer, M.-O., Khasainov, B. A. (2009). Veyssiere, B., Investigation of the behavior of solid particles dispersed by high explosive, Int. Annu. Conf. ICT 40th, 35/1-35/12.

27. Kudryashova, O. B., Vorozhtsov, B. I., Muravlev, E. V., Akhmadeev, I. R., Pavlenko, A. A. and Titov, S. S. (2011). Physicomathematical Modeling of Explosive Dispersion of Liquid and Powders, Propellants, Explos., Pyrotech. 36, 524-530.

28. Cabalo, J., Schmidt, J., Wendt, J. O. L. and Scheeline, A. (2002). Spectrometric System for Characterizing Drop and Powder Trajectories and Chemistry in Reactive Flows Applied Spectroscopy, 56, pp. 1346-1353.

29. Cabalo, J. B.; Kesavan, J.; Sickenberger, D. W.; Diviacchi, G.; Maldonado-Figueroa, C.; McGrady, D.; Stafford, K. (2016). Assessing the Biological Threat Posed by Suicide Bombers. Report ECBC-TR-1363.

30. Russell, B. D.,Benner, C. I. and Wischkaemper, J. A. (2012). Distribution feeder caused wildfires: mechanisms and prevention, Prot. Relay Eng. 43.

31. Blackburn, T. (1985) Conductor Clashing Characteristics of Overhead Lines, in: Proceedings of Electrical Energy Conferences, p.202.

32. Pagni, P. J. (1993). Causes of the 20 October 1991 Oakland-hills Conflagration, Fire Saf. J. 21, 331-339.

33. Gilbert, M. California Department of Forestry and Fire Protection Investigation Report: Incident number 07- CA-MVU-10432, http://www.fire.ca.gov/fire

34. Maranghides, A. and Mell, W. (2009) A Case Study of a Community Affected by the Witch and Guejito Fires, NIST Technical Note 1635.
http : *//fire.nist.gov/bfrlpubs/fire*09/*art*028.*html*.

35. Ramljak, I., Majstrovic, M. and Sutloviec, S. (2014). Statistical Analysis of Particles of Conductor Clashing, Dubrovnic May 13-16, 2014)

36. Zohdi, T. I. (2018). Electrodynamic machine-learning-enhanced fault-tolerance of robotic free-form printing of complex mixtures. Computational Mechanics. https://doi.org/10.1007/s00466-018-1629-y

37. Holland, J. H. 1975. *Adaptation in natural & artificial systems.* Ann Arbor, Mich. University of Michigan Press.

38. Holland, J.H.; Miller, J.H. (1991). Artificial Adaptive Agents in Economic Theory (PDF). American Economic Review. 81 (2): 365-71. Archived from the original (PDF) on October 27, 2005.

39. Goldberg, D. E. 1989. *Genetic algorithms in search, optimization & machine learning.* Addison-Wesley.

40. Davis, L. 1991. *Handbook of Genetic Algorithms.* Thompson Computer Press.

41. Onwubiko, C. 2000 *Introduction to engineering design optimization.* Prentice Hall.

42. Goldberg, D. E. & Deb, K. 2000. Special issue on Genetic Algorithms. *Computer Methods in Applied Mechanics & Engineering.* 186 (2-4) 121-124.

43. Zohdi, T. I. (2009) Mechanistic modeling of swarms. *Computer Methods in Applied Mechanics and Engineering.* Volume 198, Issues 21-26, Pages 2039-2051.

44. Zohdi, T. I. (2018). Multiple UAVs for Mapping: a review of basic modeling, simulation and applications. Annual Review of Environment and Resources. https://doi.org/10.1146/annurev-environ-102017-025912

45. Zohdi, T. I. (2019). The Game of Drones: rapid agent-based machine-learning models for multi-UAV path planning. Computational Mechanics. https://doi.org/10.1007/s00466-019-01761-9

46. Luenberger, D. 1974. *Introduction to Linear & Nonlinear Programming.* Addison-Wesley, Menlo Park.

47. Gill, P. Murray, W. and Wright, M. 1995. Practical optimization. Academic Press.

48. Zohdi, T. I. (2010) On the dynamics of charged electromagnetic particulate jets. *Archives of Computational Methods in Engineering.* Volume 17, Number 2, 109-135

49. Zohdi, T. I. (2019). The Game of Drones: rapid agent-based machine-learning models for multi-UAV path planning. Computational Mechanics. https://doi.org/10.1007/s00466-019-01761-9