function [g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params()

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%% Description:

% User defines Parameter values using this function like a script.

%

%% Inputs:

% NONE

%

%% Outputs:

% g acceleration due to gravity (m/s^2)

%

% rho density of water at 25 deg-C (kg/m^3)

% mu dynamic viscosity of water at 25 deg-C (N\*s/m^2)

%

% rho\_load density of payload material (kg/m^3)

% rho\_fins density of fins material (kg/m^3)

% rho\_hull density of hull material (kg/m^3)

% Sy\_hull tensile yield strength of hull material (N/m^2)

%

% v speed of the AUV (m/s)

% depth distance below surface of the water (m)

% T thrust from the AUV's propeller (N)

%

% theta angle of conic tail section of AUV (deg)

% n total number of helical ribs (n/a - unitless)

% alpha angle of attack of fins (deg)

% tfins thickness of hollow fins (m)

% l length of fins (m)

% w width of fins (m)

%

%% gravity

g = 9.81; % (m/s^2)

%% properties of water

rho = 1000; % (kg/m^3)

mu = 8.9\*10^-3; % (N\*s/m^2)

%% properties of AUV materials

rho\_load = 2070; % (kg/m^3)

rho\_fins = 1430; % (kg/m^3) - carbon fiber

% % Al-5058 - (http://asm.matweb.com/)

% rho\_hull = 2660; % (kg/m^3)

% Sy\_hull = 228\*10^6; % (N/m^2)

% HY-80 Steel - (http://www.matweb.com/)

rho\_hull = 7750; % (kg/m^3)

Sy\_hull = 552\*10^6; % (N/m^2)

%% AUV desired specs

v = 2; % (m/s)

depth = 1000; % (m)

%% AUV hardware

T = 34.82; % (N) <-- Blue Robotics T200 Thruster

%% AUV structural constants

theta = 30; % (deg)

alpha = 5; % (deg)

tfins = 0.01; % (m)

l = 1; % (m)

w = 1; % (m)

end

function [d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS, F\_D\_lim, V\_i\_lim] = set\_Lims()

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%% Description:

% User defines Limit values using this function like a script.

%

%% Inputs:

% NONE

%

%% Outputs:

% d\_L lower bound for AUV inner diameter (m)

% d\_U upper bound for AUV inner diameter (m)

% t\_L lower bound for AUV hull thickness (m)

% t\_U upper bound for AUV hull thickness (m)

% L\_L lower bound for AUV length (m)

% L\_U upper bound for AUV length (m)

%

% W\_lim upper limit for AUV weight (N)

% FS factor of safety (n/a - unitless)

%

%% basic variable bounds

% inner-diameter bounds

d\_L = 0.2; % (m)

d\_U = 0.5; % (m)

% thickness bounds

t\_L = 0.01; % (m)

t\_U = 0.05; % (m)

% length bounds

L\_L = 1; % (m)

L\_U = 5; % (m)

%% constraint bounds

% upper limit weight constraint

W\_lim = 7357.5; % (N)

% upper limit stress constraint

FS = 6; % (n/a - unitless)

end

function [L\_T] = calc\_L\_T(theta, D, L)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the total length of the AUV (L\_T) given the variables D, L, and theta

%

%% Inputs:

% theta angle of the conic tail section of the hull (deg)

%

% D outer diameter of the hull (m)

% L length of the cylindrical section of the hull (m)

%

%% Outputs:

% L\_T total length of the hull (m)

%

%% calculation

L\_T = D/2 + L + (D/2)\*(1/tand(theta));

end

function [S] = calc\_S(theta, D, L)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the surface area of the AUV (S) given the variables D, L, and theta

%

%% Inputs:

% theta angle of the conic tail section of the hull (deg)

%

% D outer diameter of the hull (m)

% L length of the cylindrical section of the hull (m)

%

%% Outputs:

% S surface area of the hull (m^2)

%% calculation

S = (1/2)\*pi\*D^2 + pi\*D\*L + (1/4)\*pi\*D^2\*abs(1/sind(theta));

end

function [V] = calc\_V(theta, D, L)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the volume of water displaced by the AUV (V) given the variables D, L, and theta

%

%% Inputs:

% theta angle of the conic tail section of the hull (deg)

%

% D outer diameter of the hull (m)

% L length of the cylindrical section of the hull (m)

%

%% Outputs:

% V volume of water displaced by the hull (m^3)

%

%% calculation

V = (1/12)\*pi\*D^3 + (1/4)\*pi\*D^2\*L + (1/24)\*pi\*D^3\*(1/tand(theta));

end

function [Rn] = calc\_Rn(rho, mu, v, L\_T)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the Reynolds number (Rn) given the AUV's total length and velocity

%

%% Inputs:

% rho density of water at 25 deg-C (kg/m^3)

% mu dynamic viscosity of water at 25 deg-C (N\*s/m^2)

% v speed of the AUV (m/s)

% L\_T total length of the hull (m)

%

%% Outputs:

% Rn Reynold's Number (n/a - unitless)

%

%% calculation

Rn = (rho\*v\*L\_T)/mu;

end

function [C\_f] = calc\_C\_f(Rn)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the coefficient of drag of the hull of the AUV

% due to skin-friction (C\_f) given the Reynold's Number

%

%% Inputs:

% Rn Reynold's Number (n/a - unitless)

%

%% Outputs:

% C\_f coefficient of drag of the hull due to skin-friction (n/a - unitless)

%

%% calculation

C\_f = 0.0075/(log10(Rn)-2)^2;

end

function [C\_p] = calc\_C\_p(D, L\_T, V)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the coefficient of drag of the hull of the AUV

% due to pressure loss (form factor) (C\_p) given some geometric variables

%

%% Inputs:

% D outer diameter of the hull (m)

% L\_T total length of the hull (m)

% V volume of water displaced by the hull (m^3)

%

%% Outputs:

% C\_p coefficient of drag of the hull due to skin-friction (n/a - unitless)

%

%% calculation

C\_p = V/(pi\*(D/2)^2\*L\_T);

end

function [C\_v] = calc\_C\_v(D, L\_T, C\_f, C\_p)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the total coefficient of drag of the hull of the AUV (C\_v)

% given some geometric variables, and the component coefficients of drag

% due to skin friction and pressure losses (C\_f, and C\_v)

% \*using the MIT method\*

%

%% Inputs:

% D outer diameter of the hull (m)

% L\_T total length of the hull (m)

% C\_f coefficient of drag of the hull due to skin-friction (n/a - unitless)

% C\_p coefficient of drag of the hull due to skin-friction (n/a - unitless)

%

%% Outputs:

% C\_v total coefficient of drag of the hull (n/a - unitless)

%

%% calculation

C\_v = C\_f\*(1 + (3/2)\*(D/L\_T)^(3/2) + 7\*(D/L\_T)^3 + 0.0002\*(C\_p - 0.6));

end

function [C\_v] = calc\_HydroCoeff(D, L\_T, V, Rn)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the total coefficient of drag of the hull of the AUV (C\_v)

% based on geometric quantities and Reynolds Number (Rn)

%

%% Inputs:

% D outer diameter of the hull (m)

% L\_T total length of the hull (m)

% V volume of water displaced by the hull (m^3)

% Rn Reynold's Number (n/a - unitless)

%

%% Outputs:

% C\_v total coefficient of drag of the hull (n/a - unitless)

%

%% calculation

C\_f = calc\_C\_f(Rn); % coefficient of drag due to skin-friction (n/a - unitless)

C\_p = calc\_C\_p(D, L\_T, V); % coefficient of drag due to pressure loss/form factor (n/a - unitless)

C\_v = calc\_C\_v(D, L\_T, C\_f, C\_p); % total coefficient of drag (n/a - unitless)

end

function [V\_hull] = calc\_V\_hull(theta, D, d, L)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the mass-volume of the AUV (V\_hull) given d, D, L, and theta

%

%% Inputs:

% theta angle of the conic tail section of the hull (deg)

%

% D outer diameter of the hull (m)

% d inner diameter of the hull (m)

% L length of the cylindrical section of the hull (m)

%

%% Outputs:

% V\_hull volume of the hull material (m^3)

%

%% calculation

V\_hemisphere = (1/12)\*pi\*(D^3 - d^3); % volume of front hemisphere

V\_cylinder = (1/4)\*pi\*(D^2 - d^2)\*L; % volume of middle cylinder

V\_cone = (1/24)\*pi\*(D^3 - d^3)\*(1/tand(theta)); % volume of conic tail

V\_hull = V\_hemisphere + V\_cylinder + V\_cone; % sum of all

end

function [V\_fins] = calc\_V\_fins(l, w)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the volume of water displaced by the fins

%

%% Inputs:

% l length of fins (m)

% w width of fins (m)

%

%% Outputs:

% V\_fins volume of water displaced by the fins (m^3)

%

%% calculation

cross\_section = (pi/4)\*l\*(0.12\*l); % approx. as ellipse (NACA 0012 means 0.12 height to length ration)

V\_fins = 2\*(cross\_section\*w);

End

function [V\_fins2] = calc\_V\_fins2(tfins, l, w)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the mass-volume of the hollow fins

%

%% Inputs:

% tfins thickness of fins material (m)

%

% l length of fins (m)

% w width of fins (m)

%

%% Outputs:

% V\_fins2 volume of water displaced by the fins (m^3)

%

%% calculation

cross\_section = (pi/4)\*((0.12\*l)\*l - (0.12-2\*tfins)\*l\*(l-2\*tfins));

end\_cap = (pi/4)\*l\*(0.12\*l)\*tfins;

V\_fins2 = 2\*cross\_section\*(w-tfins) + 2\*end\_cap; % http://structx.com/Shape\_Formulas\_035.html

end

function [s\_h] = calc\_s\_h(D, d, P)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the hoop stress in the cylindrical middle section of the AUV

% \*Using the thick-wall full pressure vessel equation\*

%

%% Inputs:

% D outer diameter of the hull (m)

% d inner diameter of the hull (m)

% P hydrostatic pressure (N)

%

%% Outputs:

% s\_h max hoop-stress in a cylinder (N)

%

%% calculation

s\_h = P\*D^2\*2/(D^2-d^2);

end

function [s\_a] = calc\_s\_a(D, d, P)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the axial stress in the cylindrical middle section of the AUV

% \*Using the thick-wall full pressure vessel equation\*

%

%% Inputs:

% D outer diameter of the hull (m)

% d inner diameter of the hull (m)

% P hydrostatic pressure (N)

%

%% Outputs:

% s\_a max axial stress in a cylinder (N)

%

%% calculation

s\_a = P\*d^2/(D^2 - d^2);

end

function [s\_cylinder] = calc\_s\_cylinder(s\_h, s\_a)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the Von-Mises combined stresses in the cylindrical middle section of the AUV hull

% based on the component stresses

%

%% Inputs:

% s\_h max hoop-stress in a cylinder (N)

% s\_a max axial stress in a cylinder (N)

%

%% Outputs:

% s\_cylinder Von-Mises Stress in the cylindrical section of the hull (N)

%

%% calculation

s\_cylinder = calc\_vonMises(s\_h, s\_a, 0);

end

function [s\_t] = calc\_s\_t(D, d, P)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the tangential stress in the hemispherical front section of the AUV

% \*Using the thick-wall full pressure vessel equation\*

%

%% Inputs:

% D outer diameter of the hull (m)

% d inner diameter of the hull (m)

% P hydrostatic pressure (N)

%

%% Outputs:

% s\_t max tangential stresses in a sphere (N)

%

%% calculation

s\_t = P\*D^2\*2/(D^2-d^2);

end

function [s\_sphere] = calc\_s\_sphere(s\_t)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the Von-Mises combined stresses in the spherical front section of the hull

% based on the component stresses

%

%% Inputs:

% s\_t max tangential stresses in a sphere (N)

%

%% Outputs:

% s\_sphere Von-Mises Stress in the spherical front end of the hull (N)

%

%% calculation

s\_sphere = calc\_vonMises(s\_t, s\_t, 0);

end

function [s\_H] = calc\_s\_HH(theta, D, d, P)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the hoop stress in the conical tail section of the AUV

% \*Using the thin-wall approximate pressure vessel equation\*

%

%% Inputs:

% theta angle of the conic tail section of the hull (deg)

%

% D outer diameter of the hull (m)

% d inner diameter of the hull (m)

% P hydrostatic pressure (N)

%

%% Outputs:

% s\_H max hoop stress in a cone (N)

%

%% calculation

s\_H = P\*D\*sind(theta)\*tand(theta)/(D - d);

end

function [s\_AA] = calc\_s\_AA(D, d, theta, P)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the axial stress in the conical tail section of the AUV

% \*Using the thin-wall approximate pressure vessel equation\*

%

%% Inputs:

% D outer diameter of the hull (m)

% d inner diameter of the hull (m)

% theta angle of the conic tail section of the hull (deg)

% P hydrostatic pressure (N)

%

%% Outputs:

% s\_AA max axial stress in a cone (N)

%

%% calculation

s\_AA = P\*(1/2)\*D\*sind(theta)\*tand(theta)/(D - d);

end

function [s\_cone] = calc\_s\_cone(s\_HH, s\_AA)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the Von-Mises combined stresses in the conical tail section of the AUV hull

% based on the component stresses

%

%% Inputs:

% s\_HH max hoop stress in a cone (N)

% s\_AA max axial stress in a cone (N)

%

%% Outputs:

% s\_cone Von-Mises Stress in the conical tail section of the hull (N)

%

%% calculation

s\_cone = calc\_vonMises(s\_HH, s\_AA, 0);

end

function [s\_t, s\_h, s\_a, s\_HH, s\_AA] = calc\_PV\_Stresses(theta, d, D, P)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the pressure-vessel component stresses in the AUV hull based

% on the basic geometric properties, and of course hydrostatic pressure

% \*Using the thick-wall full pressure vessel equation\*

%

%% Inputs:

% theta angle of the conic tail section of the hull (deg)

%

% d inner diameter of the hull (m)

% D outer diameter of the hull (m)

% P hydrostatic pressure (N)

%

%% Outputs:

% s\_t max tangential stresses in a sphere (N)

% s\_h max hoop-stress in a cylinder (N)

% s\_a max axial stress in a cylinder (N)

% s\_HH max hoop stress in a cone (N)

% s\_AA max axial stress in a cone (N)

%% calculation

% front hemispherical section

s\_t = calc\_s\_t(D, d, P);

% middle cylindrical section

s\_h = calc\_s\_h(D, d, P);

s\_a = calc\_s\_a(D, d, P);

% conic tail section

s\_HH = calc\_s\_HH(D, d, theta, P);

s\_AA = calc\_s\_AA(D, d, theta, P);

end

function [s\_vonMises] = calc\_vonMises(s\_1, s\_2, s\_3)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the Von-Mises combined stresses

% based on the principle stresses

%

%% Inputs:

% s\_1 1st principle stress (N/m^2)

% s\_2 2nd principle stress (N/m^2)

% s\_3 3rd principle stress (N/m^2)

%

%% Outputs:

% s\_vonMises Von-Mises Max Stress(N/m^2)

%

%% calculation

s\_vonMises = (1/sqrt(2))\*sqrt((s\_1 - s\_2)^2 + (s\_1 - s\_3)^2 + (s\_3 - s\_2)^2);

end

function [s\_max] = calc\_s\_max(s\_t, s\_h, s\_a, s\_HH, s\_AA)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the Von-Mises combined stresses in the AUV sections

% ... and the maximum throughout the entire AUV hull

% based on the component stresses

%

%% Inputs:

% s\_t max tangential stresses in a sphere (N)

% s\_h max hoop-stress in a cylinder (N)

% s\_a max axial stress in a cylinder (N)

% s\_HH max hoop stress in a cone (N)

% s\_AA max axial stress in a cone (N)

%

%% Outputs:

% s\_sphere Von-Mises Stress in the spherical front end of the hull (N)

% s\_cylinder Von-Mises Stress in the cylindrical section of the hull (N)

% s\_cone Von-Mises Stress in the conical tail section of the hull (N)

% s\_max maximum Von-Mises stress in the entire AUV hull (N)

%

%% calculation

% front hemispherical section

s\_sphere = calc\_s\_sphere(s\_t);

% middle cylindrical section

s\_cylinder = calc\_s\_cylinder(s\_h, s\_a);

% conic tail section

s\_cone = calc\_s\_cone(s\_HH, s\_AA);

% max of all sections

s\_max = max([s\_sphere, s\_cylinder, s\_cone]);

end

function [f1] = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% evaluates the objective function f1

% Force of Drag on the AUV

%

%% Inputs:

% \*parameters\*

% rho density of water at 25 deg-C (kg/m^3)

% mu dynamic viscosity of water at 25 deg-C (N\*s/m^2)

% v speed of the AUV (m/s)

% theta angle of conic tail section of AUV (deg)

% alpha angle of attack of fins (deg)

% l length of fins (m)

% w width of fins (m)

% \*variables\*

% d inner diameter of the hull (m)

% t thickness of the hull (m)

% L length of the cylindrical section of the hull (m)

%

%% Outputs:

% f1 Force of Drag on the AUV (N)

%

%% preliminary calculations

% outer diameter

D = d + 2\*t;

% total length

L\_T = calc\_L\_T(theta, D, L);

% volume displaced

V = calc\_V(theta, D, L);

% surface area of hull

S = calc\_S(theta, D, L);

% cross-section area of fins

A = 2\*w\*l;

% Reynold’s number

Rn = calc\_Rn(rho, mu, v, L\_T);

% Drag and Lift Coefficients (n/a - unitless)

[C\_v] = calc\_HydroCoeff(D, L\_T, V, Rn);

% Drag and Lift Coefficients for fins (n/a - unitless)

C\_D\_fins = 0.00012\*alpha^2 + 0.0004\*alpha + 0.0006;

% force of drag on the hull (N)

F\_D = C\_v\*(rho/2)\*S\*v^2;

% force of drag on the fins (N)

F\_D\_fins = C\_D\_fins\*(rho/2)\*A\*v^2;

% total force of drag on the AUV (N)

F\_D = F\_D + F\_D\_fins;

%% objective function

f1 = F\_D;

end

function [f2] = eval\_f2(theta, d, L)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% evaluates the objective function f2

% Internal Volume of the hull

%

%% Inputs:

% \*variables\*

% d inner diameter of the hull (m)

% L length of the hull (m)

%

%% Outputs:

% f2 Internal Volume of Cylindrical Section of the hull (m^3)

%

%% objective function

f2 = calc\_V(theta, d, L);

end

function [g7, g8, g9, g10] = eval\_g710(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, theta, alpha, tfins, l, w,...

d, t, L, ...

W\_lim, FS)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% evaluates the inequality constraints g7, g8, g9, and g10

% which are not simple upper and lower bounds on the variables

% no need to evaluate those in here for fmincon which has separate inputs

% for variable bounds (LB, UB)

%

%% Inputs:

% \*parameters\*

% g acceleration due to gravity (m/s^2)

% rho density of water at 25 deg-C (kg/m^3)

% rho\_load density of payload material (kg/m^3)

% rho\_hull density of hull material (kg/m^3)

% rho\_fins density of fins material (kg/m^3)

% Sy\_hull tensile yield strength of hull material (N/m^2)

% v speed of the AUV (m/s)

% depth distance below surface of the water (m)

% theta angle of conic tail section of AUV (deg)

% alpha angle of attack of fins (deg)

% tfins thickness of hollow fins (m)

% l length of fins (m)

% w width of fins (m)

% \*variables\*

% d inner diameter of the hull (m)

% t thickness of the hull (m)

% L length of the cylindrical section of the hull (m)

%

% \*constraint limit\*

% W\_lim upper limit for AUV weight (N)

% FS factor of safety (n/a - unitless)

%

%% Outputs:

% g7 upper bound weight constraint

% g8 lower bound buoyancy constraint

% g9 upper bound buoyancy constraint

% g10 upper bound stress constraint

%

%% preliminary calculations

% outer diameter

D = d + 2\*t;

% volume displaced

V = calc\_V(theta, D, L);

% volume displaced by fins

V\_fins = calc\_V\_fins(l, w);

% cross-section area of fins

A = 2\*w\*l;

% buoyancy

F\_B = rho\*g\*(V + V\_fins);

% Lift Coefficient for fins

C\_L\_fins = 0.1089\*alpha + 0.0194;

% force of lift from fins

F\_L = C\_L\_fins\*(rho/2)\*A\*v^2;

% mass-volume of hull

V\_hull = calc\_V\_hull(theta, D, d, L);

% mass-volume of fins

V\_fins2 = calc\_V\_fins2(tfins, l, w);

% weight

W\_hull = g\*rho\_hull\*V\_hull;

W\_fins = g\*rho\_fins\*V\_fins2;

W = W\_hull+ W\_fins;

% Internal Volume of Cylindrical Section of the hull (m^3)

V\_i = calc\_V(theta, d, L);

% resulting payload weight

W\_load = g\*rho\_load\*V\_i;

% stress

P = rho\*g\*depth; % hydrostatic pressure

[s\_t, s\_h, s\_a, s\_HH, s\_AA] = calc\_PV\_Stresses(theta, d, D, P);

s\_max = calc\_s\_max(s\_t, s\_h, s\_a, s\_HH, s\_AA);

%% inequality constraints

% upper limit weight constraint

% g7 = W + W\_load - W\_lim;

g7 = (W + W\_load)/W\_lim - 1; % normalized

% net buoyancy bounds

% g8 = (W + W\_load) - F\_B - F\_L;

% g9 = F\_B - (W + W\_load) - F\_L;

g8 = (W + W\_load - F\_B)/F\_L - 1; % normalized

g9 = (F\_B - W - W\_load)/F\_L - 1; % normalized

% upper limit stress constraint

% g10 = s\_max - (1/FS)\*Sy\_hull;

g10 = s\_max/((1/FS)\*Sy\_hull) - 1; % normalized

end

function [g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = ...

eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% evaluates all the inequality constraints values

%

%% Inputs:

% \*parameters\*

% g acceleration due to gravity (m/s^2)

% rho density of water at 25 deg-C (kg/m^3)

% rho\_load density of payload material (kg/m^3)

% rho\_fins density of fins material (kg/m^3)

% rho\_hull density of hull material (kg/m^3)

% Sy\_hull tensile yield strength of hull material (N/m^2)

% v speed of the AUV (m/s)

% depth distance below surface of the water (m)

% theta angle of conic tail section of AUV (deg)

% alpha angle of attack of fins (deg)

% tfins thickness of hollow fins (m)

% l length of fins (m)

% w width of fins (m)

% \*variables\*

% d inner diameter of the hull (m)

% t thickness of the hull (m)

% L length of the cylindrical section of the hull (m)

% \*constraint limit\*

% d\_L lower bound for AUV inner diameter (m)

% d\_U upper bound for AUV inner diameter (m)

% t\_L lower bound for AUV hull thickness (m)

% t\_U upper bound for AUV hull thickness (m)

% L\_L lower bound for AUV length (m)

% L\_U upper bound for AUV length (m)

% W\_lim upper limit for AUV weight (N)

% FS factor of safety (n/a - unitless)

%

%% Outputs:

% g1 inner-diameter lower bound constraint

% g2 inner-diameter upper bound constraint

% g3 thickness lower bound constraint

% g4 thickness upper bound constraint

% g5 length lower bound constraint

% g6 length upper bound constraint

% g7 upper bound weight constraint

% g8 lower bound buoyancy constraint

% g9 upper bound buoyancy constraint

% g10 upper bound stress constraint

%

%% preliminary calculations

% outer diameter

D = d + 2\*t;

% volume displaced

V = calc\_V(theta, D, L);

% volume displaced by fins

V\_fins = calc\_V\_fins(l, w);

% cross-section area of fins

A = 2\*w\*l;

% buoyancy

F\_B = rho\*g\*(V + V\_fins);

% Lift Coefficient for fins

C\_L\_fins = 0.1089\*alpha + 0.0194;

% force of lift from fins

F\_L = C\_L\_fins\*(rho/2)\*A\*v^2;

% mass-volume of hull

V\_hull = calc\_V\_hull(theta, D, d, L);

% mass-volume of fins

V\_fins2 = calc\_V\_fins2(tfins, l, w);

% weight

W\_hull = g\*rho\_hull\*V\_hull;

W\_fins = g\*rho\_fins\*V\_fins2;

W = W\_hull + W\_fins;

% Internal Volume of Cylindrical Section of the hull (m^3)

V\_i = calc\_V(theta, d, L);

% resulting payload weight

W\_load = g\*rho\_load\*V\_i;

% stress

P = rho\*g\*depth; % hydrostatic pressure

[s\_t, s\_h, s\_a, s\_HH, s\_AA] = calc\_PV\_Stresses(theta, d, D, P);

s\_max = calc\_s\_max(s\_t, s\_h, s\_a, s\_HH, s\_AA);

%% inequality constraints

% inner-diameter bounds

% g1 = d\_L - d;

% g2 = d - d\_U;

g1 = 1 - d/d\_L; % normalized

g2 = d/d\_U - 1; % normalized

% thickness bounds

% g3 = t\_L - t;

% g4 = t - t\_U;

g3 = 1 - t/t\_L; % normalized

g4 = t/t\_U - 1; % normalized

% length bounds

% g5 = L\_L - L;

% g6 = L - L\_U;

g5 = 1 - L/L\_L; % normalized

g6 = L/L\_U - 1; % normalized

% upper limit weight constraint

% g7 = W + W\_load - W\_lim;

g7 = (W + W\_load)/W\_lim - 1; % normalized

% net buoyancy bounds

% g8 = (W + W\_load) - F\_B - F\_L;

% g9 = F\_B - (W + W\_load) - F\_L;

g8 = (W + W\_load - F\_B)/F\_L - 1; % normalized

g9 = (F\_B - W - W\_load)/F\_L - 1; % normalized

% upper limit stress constraint

% g10 = s\_max - (1/FS)\*Sy\_hull;

g10 = s\_max/((1/FS)\*Sy\_hull) - 1; % normalized

end

function [g11] = eval\_g11(d, L, f2\_g, f2\_b, epsilon)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% evaluates the inequality constraint value g11

% lower limit internal volume constraint (epsilon constrained / minimizing drag)

%

%% Inputs:

% \*variables\*

% d inner diameter of the hull (m)

% L length of the cylindrical section of the hull (m)

%

% \*constraint limit\*

% f2\_g "good value" for internal volume (m^3)

% f2\_b "bad value" for internal volume (M^3)

% epsilon constraint level for volume, as a value between 0 and 1 for

% the normalized volume

%

%% Outputs:

% g11 lower bound internal volume constraint (<= 0)

%

%% preliminary

[~, ~, ~, ~, ~, ~, ~, ~, ~, ~, theta, ~, ~, ~, ~] = set\_Params();

%% objective function

f2 = eval\_f2(theta, d, L);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g); % normalized

%% inequality constraint

g11 = f2\_s/epsilon - 1; % normalized

end

function [g12] = eval\_g12(rho, mu, v, theta, alpha, l, w, d, t, L, f1\_g, f1\_b, epsilon)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% evaluates the inequality constraint value g12

% upper limit drag constraint (epsilon constrained / maximizing internal volume)

%

%% Inputs:

% \*parameters\*

% rho density of water at 25 deg-C (kg/m^3)

% mu dynamic viscosity of water at 25 deg-C (N\*s/m^2)

% v speed of the AUV (m/s)

% theta angle of conic tail section of AUV (deg)

% alpha angle of attack of fins (deg)

% tfins thickness of hollow fins (m)

% \*variables\*

% d inner diameter of the hull (m)

% t thickness of the hull (m)

% L length of the cylindrical section of the hull (m)

% l length of fins (m)

% w width of fins (m)

% \*constraint limit\*

% f1\_g "good value" for drag (N)

% f1\_b "bad value" for drag (N)

% epsilon constraint level for drag, as a value between 0 and 1 for

% the normalized drag

%

%% Outputs:

% g12 upper bound drag constraint (<= 0)

%

%% preliminary calculations

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g); % normalized

%% inequality constraint

g12 = f1\_s/epsilon - 1; % normalized

end

function [W, F\_B, s\_max, F\_D, V\_i, W\_load, F\_L] = calc\_Everything(d, t, L)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the objective functions and the final values related to the

% constraints other than the variable bounds. This is used for the

% visualization of the feasible region.

%

%% Inputs:

% d inner diameter of the hull (m)

% t thickness of the hull (m)

% L length of the cylindrical section of the hull (m)

%

%% Outputs:

% W weight of the empty AUV (N)

% F\_B buoyant force on the AUV (N)

% s\_max stress at max depth (N/m^2)

% F\_D force of drag on the AUV (N)

% V\_i internal volume of the cylindrical middle section of the AUV (m^3)

% W\_load weight of payload (N)

% F\_L force of lift generated by fins (N)

%

%% get all parameters

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, ~, v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

%% calculations

% outer diameter

D = d + 2\*t;

% total length

L\_T = calc\_L\_T(theta, D, L);

% volume displaced by hull

V = calc\_V(theta, D, L);

% volume displaced by fins

V\_fins = calc\_V\_fins(l, w);

% surface area of hull

S = calc\_S(theta, D, L);

% cross-section area of fins

A = 2\*w\*l;

% mass-volume of hull

V\_hull = calc\_V\_hull(theta, D, d, L);

% mass-volume of fins

V\_fins2 = calc\_V\_fins2(tfins, l, w);

% weight

W\_hull = g\*rho\_hull\*V\_hull;

W\_fins = g\*rho\_fins\*V\_fins2;

W = W\_hull+ W\_fins;

% buoyancy

F\_B = rho\*g\*(V + V\_fins);

% Reynold’s number

Rn = calc\_Rn(rho, mu, v, L\_T);

% Drag Coefficient for hull (n/a - unitless)

[C\_v] = calc\_HydroCoeff(D, L\_T, V, Rn);

% Drag and Lift Coefficients for fins (n/a - unitless)

C\_D\_fins = 0.00012\*alpha^2 + 0.0004\*alpha + 0.0006;

C\_L\_fins = 0.1089\*alpha + 0.0194;

% force of lift from fins

F\_L = C\_L\_fins\*(rho/2)\*A\*v^2;

% stress

P = rho\*g\*depth; % hydrostatic pressure

[s\_t, s\_h, s\_a, s\_HH, s\_AA] = calc\_PV\_Stresses(theta, d, D, P);

s\_max = calc\_s\_max(s\_t, s\_h, s\_a, s\_HH, s\_AA);

% Force of Drag on the AUV (N)

F\_D\_hull = C\_v\*(rho/2)\*S\*v^2;

F\_D\_fins = C\_D\_fins\*(rho/2)\*A\*v^2;

F\_D = F\_D\_hull + F\_D\_fins;

% Internal Volume of Cylindrical Section of the hull (m^3)

V\_i = calc\_V(theta, d, L);

% resulting payload weight

W\_load = g\*rho\_load\*V\_i;

% % calculate max possible velocity

% v\_max = calc\_vmax(g, rho, mu, D, L\_T, V, S, A, T, C\_D\_fins, W+W\_load);

end

function [X0] = makePaternGrid(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, Nd, Nt, NL)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% makes a 3D cubic grid of points cover the full range of variable values

% with resolution of points dictated by Nd, Nt, and NL

%

%% Inputs:

% d\_L lower bound for AUV inner diameter (m)

% d\_U upper bound for AUV inner diameter (m)

% t\_L lower bound for AUV hull thickness (m)

% t\_U upper bound for AUV hull thickness (m)

% L\_L lower bound for AUV length (m)

% L\_U upper bound for AUV length (m)

% Nd number of points in d (for grid only)

% Nt number of points in t (for grid only)

% NL number of points in L (for grid only)

%

%% Outputs:

% X0 set of start points

%

%% make grid

dd = linspace(d\_L, d\_U, Nd);

tt = linspace(t\_L, t\_U, Nt);

LL = linspace(L\_L, L\_U, NL);

[ddd, ttt, LLL] = ndgrid(dd, tt, LL);

%% reformat

X0 = [ddd(:)';ttt(:)';LLL(:)']';

end

function [X0] = makePaternRND(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, N)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% samples "N" random points from a gaussian distribution

% covering the full range of variable values

%

%% Inputs:

% d\_L lower bound for AUV inner diameter (m)

% d\_U upper bound for AUV inner diameter (m)

% t\_L lower bound for AUV hull thickness (m)

% t\_U upper bound for AUV hull thickness (m)

% L\_L lower bound for AUV length (m)

% L\_U upper bound for AUV length (m)

% N number of points to generate

%

%% Outputs:

% X0 set of start points

%

%% generate points

rng(1)

X00 = rand(N,3);

%% scale points to span the variable bounds

LB = [d\_L, t\_L, L\_L];

UB = [d\_U, t\_U, L\_U];

X0 = LB + X00.\*(UB - LB);

end

function [X0] = makePaternRND\_Halton(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, N)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% samples "N" pseudo-random points from a Halton set

% covering the full range of variable values

%

%% Inputs:

% d\_L lower bound for AUV inner diameter (m)

% d\_U upper bound for AUV inner diameter (m)

% t\_L lower bound for AUV hull thickness (m)

% t\_U upper bound for AUV hull thickness (m)

% L\_L lower bound for AUV length (m)

% L\_U upper bound for AUV length (m)

% N number of points to generate

%

%% Outputs:

% X0 set of start points

%

%% generate points

h = haltonset(3);

%% scale to span variable bounds

LB = [d\_L, t\_L, L\_L];

UB = [d\_U, t\_U, L\_U];

X0 = LB + h(1:N,:).\*(UB - LB);

end

function [X0] = genX0(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, N, Nd, Nt, NL, m)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Generates a set of start points for multi-start

%

%% Inputs:

% d\_L lower bound for AUV inner diameter (m)

% d\_U upper bound for AUV inner diameter (m)

% t\_L lower bound for AUV hull thickness (m)

% t\_U upper bound for AUV hull thickness (m)

% L\_L lower bound for AUV length (m)

% L\_U upper bound for AUV length (m)

% N number of points to generate

% Nd number of points in d (for grid only)

% Nt number of points in t (for grid only)

% NL number of points in L (for grid only)

% m method of point generation

% 1 - gaussian random

% 2 - Halton pseudo-random

% 3 - grid

% 4 - sphere

%

%% Outputs:

% X0 set of start points

%

%% generate starting points

rng(1)

if m == 1

X0 = makePaternRND(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, N);

elseif m == 2

X0 = makePaternRND\_Halton(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, N);

elseif m == 3

X0 = makePaternGrid(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, Nd, Nt, NL);

else

X0 = makePaternS(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, N);

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "GridEval"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Evaluates the inequality constraints for a full 3D cubic grid of

% points spanning the variable bounds. For those points which satisfy

% all the constraints, the objective functions are also evaluated.

%

%% Instructions:

% Just hit "run". It will save the results to GRID\_results.mat

% Plots will be generated showing the feasible domain in the

% design space, criterion space, and normalized criterion space.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

%% get all parameters:

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

%% Variable Bounds:

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% generates a set of start points

m = 3;

N = 100;

Nd = N;

Nt = N;

NL = N;

% m method of point generation

% 1 - gaussian random

% 2 - Halton pseudo-random

% 3 - grid

% 4 - sphere

% N number of points to generate

% Nd number of points in d (for grid only)

% Nt number of points in t (for grid only)

% NL number of points in L (for grid only)

X0 = genX0(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, N, Nd, Nt, NL, m);

% plot3(X0(:,1), X0(:,2), X0(:,3),'.')

% axis([d\_L, d\_U, t\_L, t\_U, L\_L, L\_U])

% xlabel('d')

% ylabel('t')

% zlabel('L')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% eval loop

j = 0;

for i = 1:size(X0,1)

d = X0(i,1);

t = X0(i,2);

L = X0(i,3);

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

if g1 > 0 || g2 > 0 || g3 > 0 || g4 > 0 || g5 > 0 || g6 > 0 || g7 > 0 || g8 > 0 || g9 > 0 || g10 > 0

% constraints violated

else

j = j+1;

% state

X(j,:) = X0(i,:);

% objective

f1(j,:) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

f2(j,:) = eval\_f2(theta, d, L);

end

end

%% good and bad values

f1\_b = max(f1);

f1\_g = min(f1);

f2\_b = min(f2);

f2\_g = max(f2);

%% normalize by good and bad values

for i = 1:j

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

Lqinf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% save results

save('GRID\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'Lq1', 'Lq2', 'Lqinf')

%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script PlotGridEval

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Generates plots of the feasible domain in the design space and

% normalized criterion space where color represents how good the

% point is according to a certain metric: f1, f2, L1, L2, Linf

%

%% Instructions:

% (check that 'GRID\_results.mat' has been created and is in the

% current folder)

% Just hit "Run".

% It will generate the plots...but it will take a while...

% Currently, saving of the plots and rotating gifs is commented out.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% clean up

close all

clear

clc

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, ~, ~] = set\_Lims();

%% GRID

GRID\_res = load('GRID\_results.mat');

X = GRID\_res.X;

f1 = GRID\_res.f1;

f2 = GRID\_res.f2;

f1\_s = GRID\_res.f1\_s;

f2\_s = GRID\_res.f2\_s;

Lq1 = GRID\_res.Lq1;

Lq2 = GRID\_res.Lq2;

LqInf = GRID\_res.Lqinf;

f1\_g = min(f1);

f1\_b = max(f1);

f2\_b = min(f2);

f2\_g = max(f2);

%% design space -----------------------------------------------------------

figure;

plot3(X(:,1), X(:,2), X(:,3), 'color', [0.5, 0.5, 0.5])

axis([d\_L, d\_U, t\_L, t\_U, L\_L, L\_U])

xlabel('d')

ylabel('t')

zlabel('L')

title('Design Space')

% saveas(gcf, 'GridEval\_DesignSpace.jpg')

% % make gif

% make\_animated\_gif('clear')

% for i = 1:360

% view(-45 + i,15)

% make\_animated\_gif('snap', gcf)

% end

% make\_animated\_gif('write','GridEval\_DesignSpace', 0.1)

%% in terms of f1\_s

figure; hold on

for i = 1:max(size(f1\_s))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*f1\_s(i); % calculate color interpolated between green (0) and red (1)

plot3(X(i,1), X(i,2),X(i,3), '.','color',c)

end

xlabel('d')

ylabel('t')

zlabel('L')

title('Design Space - f1')

axis([d\_L, d\_U, t\_L, t\_U, L\_L, L\_U])

view(-45,15)

% mark the best value

[~, idx] = min(f1\_s);

X\_f1 = X(idx, :);

plot3(X\_f1(1), X\_f1(2),X\_f1(3),'\*m')

% saveas(gcf,'GridEval\_DesignSpace\_f1.jpg')

% % make gif

% make\_animated\_gif('clear')

% for i = 1:360

% view(-45 + i,15)

% make\_animated\_gif('snap', gcf)

% end

% make\_animated\_gif('write','GridEval\_DesignSpace\_f1', 0.1)

%% in terms of f2\_s

figure; hold on

for i = 1:max(size(f2\_s))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*f2\_s(i); % calculate color interpolated between green (0) and red (1)

plot3(X(i,1), X(i,2),X(i,3), '.','color',c)

end

xlabel('d')

ylabel('t')

zlabel('L')

title('Design Space - f2')

axis([d\_L, d\_U, t\_L, t\_U, L\_L, L\_U])

view(-45,15)

% mark the best value

[~, idx] = min(f2\_s);

X\_f2 = X(idx, :);

plot3(X\_f2(1), X\_f2(2),X\_f2(3),'\*m')

% saveas(gcf,'GridEval\_DesignSpace\_f2.jpg')

% % make gif

% make\_animated\_gif('clear')

% for i = 1:360

% view(-45 + i,15)

% make\_animated\_gif('snap', gcf)

% end

% make\_animated\_gif('write','GridEval\_DesignSpace\_f2', 0.1)

%% in terms of Lq-1

Lq1 = (Lq1 - min(Lq1))/(max(Lq1) - min(Lq1)); % normalize to be between 0 and 1

figure; hold on

for i = 1:max(size(Lq1))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*Lq1(i); % calculate color interpolated between green (0) and red (1)

plot3(X(i,1), X(i,2),X(i,3), '.','color',c)

end

xlabel('d')

ylabel('t')

zlabel('L')

title('Design Space - Lq1')

axis([d\_L, d\_U, t\_L, t\_U, L\_L, L\_U])

view(-45,15)

% mark the best value

[~, idx] = min(Lq1);

X\_Lq1 = X(idx, :);

plot3(X\_Lq1(1), X\_Lq1(2),X\_Lq1(3),'\*m')

% saveas(gcf,'GridEval\_DesignSpace\_Lq1.jpg')

% % make gif

% make\_animated\_gif('clear')

% for i = 1:360

% view(-45 + i,15)

% make\_animated\_gif('snap', gcf)

% end

% make\_animated\_gif('write','GridEval\_DesignSpace\_Lq1', 0.1)

%% in terms of Lq-2

Lq2 = (Lq2 - min(Lq2))/(max(Lq2) - min(Lq2)); % normalize to be between 0 and 1

figure; hold on

for i = 1:max(size(Lq2))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*Lq2(i); % calculate color interpolated between green (0) and red (1)

plot3(X(i,1), X(i,2),X(i,3), '.','color',c)

end

xlabel('d')

ylabel('t')

zlabel('L')

title('Design Space - Lq2')

axis([d\_L, d\_U, t\_L, t\_U, L\_L, L\_U])

view(-45,15)

% mark the best value

[~, idx] = min(Lq2);

X\_Lq2 = X(idx, :);

plot3(X\_Lq2(1), X\_Lq2(2),X\_Lq2(3),'\*m')

% saveas(gcf,'GridEval\_DesignSpace\_Lq2.jpg')

% % make gif

% make\_animated\_gif('clear')

% for i = 1:360

% view(-45 + i,15)

% make\_animated\_gif('snap', gcf)

% end

% make\_animated\_gif('write','GridEval\_DesignSpace\_Lq21', 0.1)

%% in terms of Lq-inf

LqInf = (LqInf - min(LqInf))/(max(LqInf) - min(LqInf)); % normalize to be between 0 and 1

figure; hold on

for i = 1:max(size(LqInf))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*LqInf(i); % calculate color interpolated between green (0) and red (1)

plot3(X(i,1), X(i,2),X(i,3), '.','color',c)

end

xlabel('d')

ylabel('t')

zlabel('L')

title('Design Space - LqInf')

axis([d\_L, d\_U, t\_L, t\_U, L\_L, L\_U])

view(-45,15)

% mark the best value

[~, idx] = min(LqInf);

X\_Lqinf = X(idx, :);

plot3(X\_Lqinf(1), X\_Lqinf(2),X\_Lqinf(3),'\*m')

% saveas(gcf,'GridEval\_DesignSpace\_LqInf.jpg')

% % make gif

% make\_animated\_gif('clear')

% for i = 1:360

% view(-45 + i,15)

% make\_animated\_gif('snap', gcf)

% end

% make\_animated\_gif('write','GridEval\_DesignSpace\_LqInf', 0.1)

%% normalized criterion space ---------------------------------------------

figure;

plot(f1\_s, f2\_s, 'color', [0.5, 0.5, 0.5])

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space')

axis([0, 1, 0, 1])

% saveas(gcf, 'GridEval\_NormalizedCriterionSpace.jpg')

%% in terms of f1\_s

figure; hold on

for i = 1:max(size(f1\_s))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*f1\_s(i); % calculate color interpolated between green (0) and red (1)

plot(f1\_s(i), f2\_s(i),X(i), '.','color',c)

end

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space - f1')

axis([0, 1, 0, 1])

view(-45,15)

% mark the best value

plot(1, 0,'\*m')

% saveas(gcf,'GridEval\_NormalizedCriterionSpace\_f1.jpg')

%% in terms of f2\_s

figure; hold on

for i = 1:max(size(f2\_s))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*f2\_s(i); % calculate color interpolated between green (0) and red (1)

plot(f1\_s(i), f2\_s(i),X(i), '.','color',c)

end

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space - f2')

axis([0, 1, 0, 1])

view(-45,15)

% mark the best value

plot(0, 1,'\*m')

% saveas(gcf,'GridEval\_NormalizedCriterionSpace\_f2.jpg')

%% in terms of Lq-1

Lq1 = (Lq1 - min(Lq1))/(max(Lq1) - min(Lq1)); % normalize to be between 0 and 1

figure; hold on

for i = 1:max(size(Lq1))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*Lq1(i); % calculate color interpolated between green (0) and red (1)

plot(f1\_s(i), f2\_s(i),X(i), '.','color',c)

end

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space - Lq1')

axis([0, 1, 0, 1])

view(-45,15)

% mark the best value

[~, idx] = min(Lq1);

f1\_Lq1 = f1(idx);

f2\_Lq1 = f2(idx);

plot(f1\_Lq1, f2\_Lq1,'\*m')

% saveas(gcf,'GridEval\_NormalizedCriterionSpace\_Lq1.jpg')

%% in terms of Lq-2

Lq2 = (Lq2 - min(Lq2))/(max(Lq2) - min(Lq2)); % normalize to be between 0 and 1

figure; hold on

for i = 1:max(size(Lq2))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*Lq2(i); % calculate color interpolated between green (0) and red (1)

plot(f1\_s(i), f2\_s(i),X(i), '.','color',c)

end

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space - Lq2')

axis([0, 1, 0, 1])

view(-45,15)

% mark the best value

[~, idx] = min(Lq2);

f1\_Lq1 = f1(idx);

f2\_Lq1 = f2(idx);

plot(f1\_Lq1, f2\_Lq1,'\*m')

% saveas(gcf,'GridEval\_NormalizedCriterionSpace\_Lq2.jpg')

%% in terms of Lq-inf

LqInf = (LqInf - min(LqInf))/(max(LqInf) - min(LqInf)); % normalize to be between 0 and 1

figure; hold on

for i = 1:max(size(LqInf))

c = [0,1,0] + ([1,0,0] - [0,1,0])\*LqInf(i); % calculate color interpolated between green (0) and red (1)

plot(f1\_s(i), f2\_s(i),X(i), '.','color',c)

end

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space - LqInf')

axis([0, 1, 0, 1])

view(-45,15)

% mark the best value

[~, idx] = min(LqInf);

f1\_Lq1 = f1(idx);

f2\_Lq1 = f2(idx);

plot(f1\_Lq1, f2\_Lq1,'\*m')

% saveas(gcf,'GridEval\_NormalizedCriterionSpace\_LqInf.jpg')

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "fminimax\_script"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description

% Uses the built-in MATLAB function "fminimax"

% for multi-objective optimization.

%

%% Instructions:

% Just hit "Run"

% It will display the results in the command window

% and it will save the results in 'fminimax\_results.mat'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% limits

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% variable bounds

LB = [d\_L, t\_L, L\_L]; % Variable Lower Bounds

UB = [d\_U, t\_U, L\_U]; % Variable Upper Bounds

% initial guess

d0 = d\_L + (d\_U - d\_L)\*(0);

t0 = t\_L + (t\_U - t\_L)\*(0);

L0 = L\_L + (L\_U - L\_L)\*(0);

X0 = [d0, t0, L0]; % known feasible point

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% optimization / minimization

% currently these options are pretty much default, but I put this here for

% potential uses in the future

options = optimoptions(@fminimax, ...

'ConstraintTolerance', 1e-6, ...

'MaxFunctionEvaluations', 2000, ...

'MaxIterations', 1500, ...

'StepTolerance', 1e-10);

% do it!

[X\_opt, f\_opt] = fminimax(@myfun, X0, [], [], [], [], LB, UB, @mycon, options);

%% minimized objective values

d = X\_opt(1);

t = X\_opt(2);

L = X\_opt(3);

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

%% display results

fprintf('\n\n Optimization Method: ')

fprintf('\n fminimax ')

fprintf('\n\n Multi-Objective Optimum: ')

fprintf('\n D = %3.3f m ', d);

fprintf('\n t = %3.3f m ', t);

fprintf('\n L = %3.3f m ', L);

fprintf('\n\n Optimized Objective:')

fprintf('\n f1 = %3.3f N', f1)

fprintf('\n f2 = %3.3f m^3', f2)

fprintf('\n\n Scaled Values: ')

fprintf('\n f1g = %3.3f N, \t f1b = %3.3f N', f1\_g, f1\_b)

fprintf('\n f2g = %3.3f m^3, \t f2b = %3.3f m^3', f2\_g, f2\_b)

fprintf('\n f1s = %3.3f', f1\_s)

fprintf('\n f2s = %3.3f', f2\_s)

fprintf('\n\n')

%% save results

save('fminimax\_results.mat', 'X\_opt', 'f1', 'f2', 'f1\_s', 'f2\_s')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function

function f = myfun(X)

[~, rho, mu, ~, ~, ~, ~, v, ~, ~, theta, alpha, ~, l, w] = set\_Params();

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% scaled values

f(1) = (f1 - f1\_g)/(f1\_b - f1\_g);

f(2) = (f2 - f2\_g)/(f2\_b - f2\_g);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% nonlinear constraint function

function [c, ceq] = mycon(X)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, ~, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[~, ~, ~, ~, ~, ~, W\_lim, FS] = set\_Lims();

% g7 = upper bound weight constraint

% g8 = lower bound buoyancy constraint

% g9 = upper bound buoyancy constraint

% g10 = upper bound stress constraint

[g7, g8, g9, g10] = eval\_g710(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w,...

d, t, L, ...

W\_lim, FS);

% inequality constraints

c = [g7, g8, g9, g10];

% equality constraints

ceq = [];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "weighted\_fimincon\_script"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Uses the built-in MATLAB function "fmincon" in conjunction with

% the Weighted Method to achieve multi-objective optimization.

% The weight for the first objective is swept from 0 to 1.

%

%% Instructions:

% Just hit "Run"

% A plot of the optima achieved in the criterion space is generated.

% A plot of the objective functions vs the weight

% assigned to the first objective are generated.

% And the results are saved in 'fmincon\_results.mat'

% Currently, saving of the plots are commented out

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% limits

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% variable bounds

LB = [d\_L, t\_L, L\_L]; % Variable Lower Bounds

UB = [d\_U, t\_U, L\_U]; % Variable Upper Bounds

% initial guess

d0 = d\_L + (d\_U - d\_L)\*(1);

t0 = t\_L + (t\_U - t\_L)\*(1);

L0 = L\_L + (L\_U - L\_L)\*(1);

X0 = [d0, t0, L0];

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% optimization / minimization

% currently these options are pretty much default, but I put this here for

% potential uses in the future

options = optimoptions(@fmincon, ...

'ConstraintTolerance', 1e-6, ...

'MaxFunctionEvaluations', 2000, ...

'MaxIterations', 1500, ...

'StepTolerance', 1e-10);

j = 0;

for i = 0:100

% multi-objective weights

w1 = i/100;

w2 = 1 - w1;

% do it!

[X\_opt, ~] = fmincon(@(X)myfun(X, w1, w2), X0, [], [], [], [], LB, UB, @mycon, options);

d = X\_opt(1);

t = X\_opt(2);

L = X\_opt(3);

% check feasibility

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

if g1 > 0 || g2 > 0 || g3 > 0 || g4 > 0 || g5 > 0 || g6 > 0 || g7 > 0 || g8 > 0 || g9 > 0 || g10 > 0

% constraints violated

else

j = j+1;

% params

W1(j,:) = w1;

W2(j,:) = w2;

% state

X(j,:) = X\_opt;

% Force of Drag (N)

f1(j,:) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2(j,:) = eval\_f2(theta, d, L);

end

end

% update good and bad

if min(f1) < f1\_g

f1\_g = min(f1);

end

if max(f1) > f1\_b

f1\_b = max(f1);

end

if min(f2) < f2\_b

f2\_b = min(f2);

end

if max(f2) > f2\_g

f2\_g = max(f2);

end

% normalize

for i = 1:j

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

Lqinf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% plots

% normalized criterion space

figure;

plot(f1\_s, f2\_s,'.b')

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space')

axis([0,1,0,1])

%saveas(gcf,'fmincon\_NormalizedCriterionSpace.jpg')

% normalized criterion vs weight

figure; hold on

plot(W1, f1\_s, '.')

plot(W1, f2\_s, '.')

xlabel('W1')

legend('f1 - drag', 'f2 - volume')

title('normalized criterion space vs weight')

axis([0,1,0,1])

% saveas(gcf,'fmincon\_NormalizedCriterionSpace\_WeightedTrend.jpg')

%% save results

save('fmincon\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'W1', 'W2', 'Lq1', 'Lq2', 'Lqinf')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function

function f = myfun(X, w1, w2)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[~, rho, mu, ~, ~, ~, ~, v, ~, ~, theta, alpha, ~, l, w] = set\_Params();

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

% weighted sum

f = w1\*f1\_s + w2\*f2\_s;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% nonlinear constraint function

function [c, ceq] = mycon(X)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, ~, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[~, ~, ~, ~, ~, ~, W\_lim, FS] = set\_Lims();

% g7 = upper bound weight constraint

% g8 = lower bound buoyancy constraint

% g9 = upper bound buoyancy constraint

% g10 = upper bound stress constraint

[g7, g8, g9, g10] = eval\_g710(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w,...

d, t, L, ...

W\_lim, FS);

% inequality constraints

c = [g7, g8, g9, g10];

% equality constraints

ceq = [];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "weighted\_PenaltyMethod"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Uses a custom-made Penalty Method

% (utilizing fminunc for unconstrained optimization)

% in conjunction with the Weighted Method to achieve multi-objective

% optimization.

% The weight for the first objective is swept from 0 to 1.

%

%% Instructions:

% Set the initial position X0, and then hit "Run"

% A plot of the optima achieved in the criterion space is generated.

% A plot of the objective functions vs the weight

% assigned to the first objective is generated.

% And the results are saved in 'Penalty\_results.mat'

% Currently, saving of the plots are commented out

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% limits

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% variable bounds

LB = [d\_L, t\_L, L\_L]; % Variable Lower Bounds

UB = [d\_U, t\_U, L\_U]; % Variable Upper Bounds

% initial guess \*\*\* set by user \*\*\*

d0 = d\_L + (d\_U - d\_L)\*(0);

t0 = t\_L + (t\_U - t\_L)\*(0);

L0 = L\_L + (L\_U - L\_L)\*(0);

X0 = [d0, t0, L0];

%% algorithm parameters

% penalty grow rate \*\*

gamma = 1.5;

% max penalty \*\*

max\_rp = 1e3;

% INFESIBILITY tolerance

tol = 1e-6;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% iterative optimization loop

j = 0;

for i = 0:100

% multi-objective weights

w1 = i/100;

w2 = 1 - w1;

% initialize penalty method

rp = 1;

INFESIBILITY = 1;

INFESIBILITY\_old = 1;

nochange = 0;

% iterative penalty method

while INFESIBILITY > tol && nochange == 0

% unconstrained minimization

[X\_min, f\_min] = fminunc(@(X)fun(X, rp, w1, w2), X0);

% constraint check

[INFESIBILITY] = con(X\_min);

% update penalty

rp = gamma\*rp;

if rp > max\_rp

rp = max\_rp;

end

% X0(i,:) = X\_min;

if INFESIBILITY == INFESIBILITY\_old

nochange = 1;

end

INFESIBILITY\_old = INFESIBILITY;

end

% save results for all multi-starts within infeasibility tolerance

if INFESIBILITY < tol

j = j + 1;

% params

W1(j,:) = w1;

W2(j,:) = w2;

% state

X(j,:) = X\_min;

d = X(j,1);

t = X(j,2);

L = X(j,3);

% Force of Drag (N)

f1(j,:) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2(j,:) = eval\_f2(theta, d, L);

end

end

% normalize

for i = 1:j

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

Lqinf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% plots

% normalized criterion space

figure;

plot(f1\_s, f2\_s,'.b')

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space')

axis([0,1,0,1])

% saveas(gcf,'penalty\_NormalizedCriterionSpace.jpg')

% normalized criterion vs weight

figure; hold on

plot(W1, f1\_s, '.')

plot(W1, f2\_s, '.')

xlabel('W1')

legend('f1 - drag', 'f2 - volume')

title('Normalized Criterion Space vs Weight')

axis([0,1,0,1])

% saveas(gcf,'penalty\_NormalizedCriterionSpace\_WeightedTrend.jpg')

%% save results

save('penalty\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'W1', 'W2', 'Lq1', 'Lq2', 'Lqinf')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function

function phi = fun(X, rp, w1, w2)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

% weighted sum

f = w1\*f1\_s + w2\*f2\_s;

% constraints

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

INFESIBILITY = ...

max(0, g1) + max(0, g2) + max(0, g3) + max(0, g4) + max(0, g5) + ...

max(0, g6) + max(0, g7) + max(0, g8) + max(0, g9) + max(0, g10);

phi = f + rp\*INFESIBILITY;

end

%% constraints reformulated as Infusibility

function [INFESIBILITY] = con(X)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, ~, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% constraints

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

INFESIBILITY = sumsqr([max(0, g1),max(0, g2),max(0, g3),max(0, g4),max(0, g5),...

max(0, g6),max(0, g7),max(0, g8),max(0, g9),max(0, g10)]);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "weighted\_PenaltyMethod"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Uses a custom-made Penalty Method

% (utilizing fminunc for unconstrained optimization)

% in conjunction with the Weighted Method to achieve multi-objective

% optimization.

% The weight for the first objective is swept from 0 to 1.

%

%% Instructions:

% Set the initial position X0, and then hit "Run"

% A plot of the optima achieved in the criterion space is generated.

% A plot of the objective functions vs the weight

% assigned to the first objective is generated.

% And the results are saved in 'Penalty\_results.mat'

% Currently, saving of the plots are commented out

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% limits

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% variable bounds

LB = [d\_L, t\_L, L\_L]; % Variable Lower Bounds

UB = [d\_U, t\_U, L\_U]; % Variable Upper Bounds

% initial guess \*\*\* set by user \*\*\*

d0 = d\_L + (d\_U - d\_L)\*(1/2);

t0 = t\_L + (t\_U - t\_L)\*(1/2);

L0 = L\_L + (L\_U - L\_L)\*(1/2);

X0 = [d0, t0, L0];

%% algorithm parameters

% penalty grow rate \*\*

gamma = 1.5;

% max penalty \*\*

max\_rp = 1e3;

% INFESIBILITY tolerance

tol = 1e-6;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% iterative optimization loop

j = 0;

for i = 0:100

% multi-objective weights

w1 = i/100;

w2 = 1 - w1;

% initialize penalty method

rp = 1;

INFESIBILITY = 1;

INFESIBILITY\_old = 1;

nochange = 0;

% iterative penalty method

while INFESIBILITY > tol && nochange == 0

% unconstrained minimization

[X\_min, f\_min] = fminunc(@(X)fun(X, rp, w1, w2), X0);

% constraint check

[INFESIBILITY] = con(X\_min);

% update penalty

rp = gamma\*rp;

if rp > max\_rp

rp = max\_rp;

end

% X0(i,:) = X\_min;

if INFESIBILITY == INFESIBILITY\_old

nochange = 1;

end

INFESIBILITY\_old = INFESIBILITY;

end

% save results for all multi-starts within infeasibility tolerance

if INFESIBILITY < tol

j = j + 1;

% params

W1(j,:) = w1;

W2(j,:) = w2;

% state

X(j,:) = X\_min;

d = X(j,1);

t = X(j,2);

L = X(j,3);

% Force of Drag (N)

f1(j,:) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2(j,:) = eval\_f2(theta, d, L);

end

end

% normalize

for i = 1:j

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

Lqinf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% plots

% normalized criterion space

figure;

plot(f1\_s, f2\_s,'.b')

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space')

axis([0,1,0,1])

% saveas(gcf,'penalty2\_NormalizedCriterionSpace.jpg')

% normalized criterion vs weight

figure; hold on

plot(W1, f1\_s, '.')

plot(W1, f2\_s, '.')

xlabel('W1')

legend('f1 - drag', 'f2 - volume')

title('Normalized Criterion Space vs Weight')

axis([0,1,0,1])

% saveas(gcf,'penalty2\_NormalizedCriterionSpace\_WeightedTrend.jpg')

%% save results

save('penalty2\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'W1', 'W2', 'Lq1', 'Lq2', 'Lqinf')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function

function phi = fun(X, rp, w1, w2)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

% weighted sum

f = w1\*f1\_s + w2\*f2\_s;

% constraints

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

INFESIBILITY = ...

max(0, g1) + max(0, g2) + max(0, g3) + max(0, g4) + max(0, g5) + ...

max(0, g6) + max(0, g7) + max(0, g8) + max(0, g9) + max(0, g10);

phi = f + rp\*INFESIBILITY;

end

%% constraints reformulated as infeasibility

function [INFESIBILITY] = con(X)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, ~, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% constraints

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

INFESIBILITY = sumsqr([max(0, g1),max(0, g2),max(0, g3),max(0, g4),max(0, g5),...

max(0, g6),max(0, g7),max(0, g8),max(0, g9),max(0, g10)]);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "MultiStart\_weighted\_PenaltyMethod"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Uses a custom-made Penalty Method

% (utilizing fminunc for unconstrained optimization)

% in conjunction with the Weighted Method to achieve multi-objective

% optimization.

% Multi-Start is also used to overcome the discrepancy between

% solutions depending on the start point. Many different starting

% points are tried, and the best solution is selected.

% The weight for the first objective is swept from 0 to 1.

% For each weight, multi-start is used.

%

%% Instructions:

% Select the parameters for the Multi-Start generator

% m method of point generation

% 1 - gaussian random

% 2 - Halton pseudo-random

% 3 - grid

% 4 - sphere

% N number of points to generate

% Nd number of points in d (for grid only)

% Nt number of points in t (for grid only)

% NL number of points in L (for grid only)

%

% Then, hit "Run"

% A plot of the optima achieved in the criterion space is generated.

% A plot of the objective functions vs the weight

% assigned to the first objective is generated.

% And the results are saved in 'multistart\_results.mat'

% Currently, saving of the plots are commented out

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% limits

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

%% algorithm parameters

% penalty grow rate \*\*

gamma = 1.5;

% max penalty \*\*

max\_rp = 1e3;

% INFESIBILITY tolerance

tol = 1e-6;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% generates a set of start points

m = 3;

N = 5;

Nd = N;

Nt = N;

NL = N;

% m method of point generation

% 1 - gaussian random

% 2 - Halton pseudo-random

% 3 - grid

% 4 - sphere

% N number of points to generate

% Nd number of points in d (for grid only)

% Nt number of points in t (for grid only)

% NL number of points in L (for grid only)

X0 = genX0(d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, N, Nd, Nt, NL, m);

% % plot multi-start points

% plot3(X0(:,1), X0(:,2), X0(:,3),'.')

% axis([d\_L, d\_U, t\_L, t\_U, L\_L, L\_U])

% xlabel('d')

% ylabel('t')

% zlabel('L')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% iterative optimization loop

jj = 0;

for ii = 0:100

% multi-objective weights

w1 = ii/100;

w2 = 1 - w1;

j = 0;

X\_multi = [];

f\_multi = [];

% multi-start

for i = 1:size(X0,1)

% check feasibility

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

X0(i,1), X0(i,2), X0(i,3), ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

% penalty method must start outside the feasible region

if g1 > 0 || g2 > 0 || g3 > 0 || g4 > 0 || g5 > 0 || g6 > 0 || g7 > 0 || g8 > 0 || g9 > 0 || g10 > 0

% initialize penalty method

rp = 1;

INFESIBILITY = 1;

INFESIBILITY\_old = 1;

nochange = 0;

% iterative penalty method

while INFESIBILITY > tol && nochange == 0

% unconstrained minimization

[X\_min, f\_min] = fminunc(@(X)fun(X, rp, w1, w2), X0(i,:));

% constraint check

[INFESIBILITY] = con(X\_min);

% update penalty

rp = gamma\*rp;

if rp > max\_rp

rp = max\_rp;

end

% X0(i,:) = X\_min;

if INFESIBILITY == INFESIBILITY\_old

nochange = 1;

end

INFESIBILITY\_old = INFESIBILITY;

end

% save results for all multi-starts within infeasibility tolerance

if INFESIBILITY < tol

j = j + 1;

X\_multi(j,:) = X\_min;

f\_multi(j,:) = f\_min;

end

end

end

% if a feasible solution was found, record it

if ~isempty(f\_multi)

% find best of multi-start for this weight

[f\_opt, idx] = min(f\_multi);

jj = jj + 1;

% params

W1(jj,:) = w1;

W2(jj,:) = w2;

% state

X(jj,:) = X\_multi(idx,:);

d = X(jj,1);

t = X(jj,2);

L = X(jj,3);

% Force of Drag (N)

f1(jj,:) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2(jj,:) = eval\_f2(theta, d, L);

end

end

% normalize

for i = 1:jj

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

Lqinf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% plots

% normalized criterion space

figure;

plot(f1\_s, f2\_s,'.b')

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space')

% saveas(gcf,'multistart\_NormalizedCriterionSpace.jpg')

% normalized criterion vs weight

figure; hold on

plot(W1, f1\_s, '.')

plot(W1, f2\_s, '.')

xlabel('W1')

legend('f1 - drag', 'f2 - volume')

title('Normalized Criterion Space vs Weight')

% saveas(gcf,'multistart\_NormalizedCriterionSpace\_WeightedTrend.jpg')

%% save results

save('multistart\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'W1', 'W2', 'Lq1', 'Lq2', 'Lqinf')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function

function phi = fun(X, rp, w1, w2)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

% weighted sum

f = w1\*f1\_s + w2\*f2\_s;

% constraints

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

INFESIBILITY = ...

max(0, g1) + max(0, g2) + max(0, g3) + max(0, g4) + max(0, g5) + ...

max(0, g6) + max(0, g7) + max(0, g8) + max(0, g9) + max(0, g10);

phi = f + rp\*INFESIBILITY;

end

%% constraints reformulated as infeasibility

function [INFESIBILITY] = con(X)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, ~, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% constraints

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

INFESIBILITY = sumsqr([max(0, g1),max(0, g2),max(0, g3),max(0, g4),max(0, g5),...

max(0, g6),max(0, g7),max(0, g8),max(0, g9),max(0, g10)]);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "weighted\_AugmentedLagrangianMethod"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Uses a custom made Augmented Lagrangian Method

% (utilizing fminunc for unconstrained optimization)

% in conjunction with the Weighted Method to achieve multi-objective

% optimization.

% The weight for the first objective is swept from 0 to 1.

%

%% Instructions:

% Set the initial position X0, and then hit "Run"

% A plot of the optima achieved in the criterion space is generated.

% A plot of the objective functions vs the weight

% assigned to the first objective is generated.

% And the results are saved in 'ALM\_results.mat'

% Currently, saving of the plots are commented out

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% limits

[g, rho, mu\_w, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% variable bounds

LB = [d\_L, t\_L, L\_L]; % Variable Lower Bounds

UB = [d\_U, t\_U, L\_U]; % Variable Upper Bounds

% initial guess

d0 = d\_L + (d\_U - d\_L)\*(0);

t0 = t\_L + (t\_U - t\_L)\*(0);

L0 = L\_L + (L\_U - L\_L)\*(0);

X0 = [d0, t0, L0];

%% algorithm parameters

% initial mu

mu = zeros(10,1);

% penalty grow rate \*\*

gamma = 1.5;

% max penalty \*\*

max\_rp = 1e3;

% INFESIBILITY tolerance

tol = 1e-6;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% iterative optimization loop

j = 0;

for i = 0:100

% multi-objective weights

w1 = i/100;

w2 = 1 - w1;

% initialize penalty method

rp = 1;

INFESIBILITY = 1;

INFESIBILITY\_old = 1;

nochange = 0;

% iterative penalty method

while INFESIBILITY > tol && nochange == 0

% unconstrained minimization

[X\_min, f\_min] = fminunc(@(X)fun(X, rp, mu, w1, w2), X0);

% constraint check& parameter update

[INFESIBILITY, rp, mu] = conup(X\_min, rp, mu, gamma);

% update penalty

if rp > max\_rp

rp = max\_rp;

end

% X0(i,:) = X\_min;

if INFESIBILITY == INFESIBILITY\_old

nochange = 1;

end

INFESIBILITY\_old = INFESIBILITY;

end

% save results for all multi-starts within infeasibility tolerance

if INFESIBILITY < tol

j = j + 1;

% params

W1(j,:) = w1;

W2(j,:) = w2;

% state

X(j,:) = X\_min;

d = X(j,1);

t = X(j,2);

L = X(j,3);

% Force of Drag (N)

f1(j,:) = eval\_f1(rho, mu\_w, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2(j,:) = eval\_f2(theta, d, L);

end

end

% normalize

for i = 1:j

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

Lqinf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% plots

% normalized criterion space

figure;

plot(f1\_s, f2\_s,'.b')

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space')

axis([0,1,0,1])

% saveas(gcf,'ALM\_NormalizedCriterionSpace.jpg')

% normalized criterion vs weight

figure; hold on

plot(W1, f1\_s, '.')

plot(W1, f2\_s, '.')

xlabel('W1')

legend('f1 - drag', 'f2 - volume')

title('Normalized Criterion Space vs Weight')

axis([0,1,0,1])

% saveas(gcf,'ALM\_NormalizedCriterionSpace\_WeightedTrend.jpg')

%% save results

save('ALM\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'W1', 'W2', 'Lq1', 'Lq2', 'Lqinf')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function

function A = fun(X, rp, mu, w1, w2)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% split mu

mu1 = mu(1);

mu2 = mu(2);

mu3 = mu(3);

mu4 = mu(4);

mu5 = mu(5);

mu6 = mu(6);

mu7 = mu(7);

mu8 = mu(8);

mu9 = mu(9);

mu10 = mu(10);

% parameters

[g, rho, mu\_w, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% Force of Drag (N)

f1 = eval\_f1(rho, mu\_w, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

% weighted sum

f = w1\*f1\_s + w2\*f2\_s;

% calc constraints

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

% calc psi

psi1 = max(-mu1/(2\*rp), g1);

psi2 = max(-mu2/(2\*rp), g2);

psi3 = max(-mu3/(2\*rp), g3);

psi4 = max(-mu4/(2\*rp), g4);

psi5 = max(-mu5/(2\*rp), g5);

psi6 = max(-mu6/(2\*rp), g6);

psi7 = max(-mu7/(2\*rp), g7);

psi8 = max(-mu8/(2\*rp), g8);

psi9 = max(-mu9/(2\*rp), g9);

psi10 = max(-mu10/(2\*rp), g10);

% calc infeasibility

INFESIBILITY = sumsqr([psi1,psi2,psi3,psi4,psi5,psi6,psi7,psi8,psi9,psi10]);

% calc Lagrangian

L = f + ...

mu1\*psi1 + mu2\*psi2 + mu3\*psi3 + mu4\*psi4 + mu5\*psi5 + ...

mu6\*psi6 + mu7\*psi7 + mu8\*psi8 + mu9\*psi9 + mu10\*psi10;

% calc augmented Lagrangian

A = L + rp\*INFESIBILITY;

end

%% constraints reformulated as infeasibility

function [INFESIBILITY, rp, mu] = conup(X, rp, mu, gamma)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% split mu

mu1 = mu(1);

mu2 = mu(2);

mu3 = mu(3);

mu4 = mu(4);

mu5 = mu(5);

mu6 = mu(6);

mu7 = mu(7);

mu8 = mu(8);

mu9 = mu(9);

mu10 = mu(10);

% parameters

[g, rho, ~, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% calc constraints

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

% calc psi

psi1 = max(-mu1/(2\*rp), g1);

psi2 = max(-mu2/(2\*rp), g2);

psi3 = max(-mu3/(2\*rp), g3);

psi4 = max(-mu4/(2\*rp), g4);

psi5 = max(-mu5/(2\*rp), g5);

psi6 = max(-mu6/(2\*rp), g6);

psi7 = max(-mu7/(2\*rp), g7);

psi8 = max(-mu8/(2\*rp), g8);

psi9 = max(-mu9/(2\*rp), g9);

psi10 = max(-mu10/(2\*rp), g10);

% calc infeasibility

INFESIBILITY = sumsqr([psi1,psi2,psi3,psi4,psi5,psi6,psi7,psi8,psi9,psi10]);

% update mu's

mu1 = mu1 + 2\*rp\*psi1;

mu2 = mu2 + 2\*rp\*psi2;

mu3 = mu3 + 2\*rp\*psi3;

mu4 = mu4 + 2\*rp\*psi4;

mu5 = mu5 + 2\*rp\*psi5;

mu6 = mu6 + 2\*rp\*psi6;

mu7 = mu7 + 2\*rp\*psi7;

mu8 = mu8 + 2\*rp\*psi8;

mu9 = mu9 + 2\*rp\*psi9;

mu10 = mu10 + 2\*rp\*psi10;

% reform mu

mu = [mu1, mu2, mu3, mu4, mu5, mu6, mu7, mu8, mu9, mu10];

% update penalty

rp = gamma\*rp;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "GlobalCriterionMethod"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% A custom Lq method multi-objective optimizer (utilizing fmincon for

% constrained single-objective optimization).

%% Instructions:

% Choose a value for q, and then hit "Run". It will print out the

% chosen q the multi-objective solution.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% \*\* q = 1, 2, inf \*\*

q = 2;

% limits

[~, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% variable bounds

LB = [d\_L, t\_L, L\_L]; % Variable Lower Bounds

UB = [d\_U, t\_U, L\_U]; % Variable Upper Bounds

% initial guess

d0 = d\_L + (d\_U - d\_L)\*(1);

t0 = t\_L + (t\_U - t\_L)\*(1);

L0 = L\_L + (L\_U - L\_L)\*(1);

X0 = [d0, t0, L0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% optimization / minimization

% currently these options are pretty much default, but I put this here for

% potential uses in the future

options = optimoptions(@fmincon, ...

'ConstraintTolerance', 1e-6, ...

'MaxFunctionEvaluations', 2000, ...

'MaxIterations', 1500, ...

'StepTolerance', 1e-10);

% single objective optimization to find the ideal point

[X\_opt1, f1\_0] = fmincon(@(X)myfun1(X), X0, [], [], [], [], LB, UB, @mycon, options);

[X\_opt2, f2\_0] = fmincon(@(X)myfun2(X), X0, [], [], [], [], LB, UB, @mycon, options);

% multi-objective optimization according to the Lq method

[X\_opt, f\_opt] = fmincon(@(X)myfunLq(X, q, f1\_0, f2\_0), X0, [], [], [], [], LB, UB, @mycon, options);

%% minimized objective values

d = X\_opt(1);

t = X\_opt(2);

L = X\_opt(3);

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

%% display results

fprintf('\n\n Optimization Method: ')

fprintf('\n Global Criterion (Lq) Method [with fmincon] ')

fprintf('\n\n Optimizer Parameters: ')

fprintf('\n q = %3.3f ', q);

fprintf('\n\n Multi-Objective Optimum: ')

fprintf('\n D = %3.3f m ', d);

fprintf('\n t = %3.3f m ', t);

fprintf('\n L = %3.3f m ', L);

fprintf('\n\n Optimized Objective:')

fprintf('\n f1 = %3.3f N', f1)

fprintf('\n f2 = %3.3f m^3', f2)

fprintf('\n\n Scaled Values: ')

fprintf('\n f1s = %3.3f', f1\_s)

fprintf('\n f2s = %3.3f', f2\_s)

fprintf('\n\n')

%% save results

save(strcat('Lq',num2str(q),'\_results.mat'), 'X\_opt', 'f1', 'f2', 'f1\_s', 'f2\_s')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function - single objective #1

function f1\_s = myfun1(X)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[~, rho, mu, ~, ~, ~, ~, v, ~, ~, theta, alpha, ~, l, w] = set\_Params();

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function - single objective #2

function f2\_s = myfun2(X)

% variables

d = X(1); % inner diameter of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% params

[~, ~, ~, ~, ~, ~, ~, ~, ~, ~, theta, ~, ~, ~, ~] = set\_Params();

% good and bad values

f2\_b = 0.0353;

f2\_g = 0.1242;

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% scaled values

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function - MULTIOBJECTIVE

function Lq = myfunLq(X, q, f1\_0, f2\_0)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[~, rho, mu\_w, ~, ~, ~, ~, ...

v, ~, ~, theta, alpha, ~, l, w] = set\_Params();

% Force of Drag (N)

f1 = eval\_f1(rho, mu\_w, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

% L-q norm

if isinf(q)

Lq = max(f1\_s - f1\_0, f2\_s - f2\_0);

else

Lq = (abs(f1\_s - f1\_0)^q + abs(f2\_s - f2\_0)^q)^(1/q);

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% nonlinear constraint function

function [c, ceq] = mycon(X)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, ~, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[~, ~, ~, ~, ~, ~, W\_lim, FS] = set\_Lims();

% g7 = upper bound weight constraint

% g8 = lower bound buoyancy constraint

% g9 = upper bound buoyancy constraint

% g10 = upper bound stress constraint

[g7, g8, g9, g10] = eval\_g710(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w,...

d, t, L, ...

W\_lim, FS);

% inequality constraints

c = [g7, g8, g9, g10];

% equality constraints

ceq = [];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "EpsilonConstrainedMethod\_OptDrag"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% A custom-made epsilon constrained method for multi-objective

% optimization by converting one of the two objectives into an

% inequality constraint.

% (utilizing fmincon for constrained single-objective optimization)

% There are two separate files for this method.

% This one is for optimizing drag while constraining volume.

%

%% Instructions:

% Just hit "Run". A plot of the objective functions vs epsilon is generated.

% And the results are saved in 'EC\_results.mat'.

% Currently, saving of the plots are commented out.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% limits

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% variable bounds

LB = [d\_L, t\_L, L\_L]; % Variable Lower Bounds

UB = [d\_U, t\_U, L\_U]; % Variable Upper Bounds

% initial guess

d0 = d\_L + (d\_U - d\_L)\*(1);

t0 = t\_L + (t\_U - t\_L)\*(1);

L0 = L\_L + (L\_U - L\_L)\*(1);

X0 = [d0, t0, L0];

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% optimization / minimization

% currently these options are pretty much default, but I put this here for

% potential uses in the future

options = optimoptions(@fmincon, ...

'ConstraintTolerance', 1e-6, ...

'MaxFunctionEvaluations', 2000, ...

'MaxIterations', 1500, ...

'StepTolerance', 1e-10);

j = 0;

for i = 1:100

% multi-objective weights

epsilon = i/100;

% optimize f1, while constraining f2 <= V\_i\_lim (set in set\_Lims)

[X\_opt, f] = fmincon(@(X)myfun(X), X0, [], [], [], [], LB, UB, @(X)mycon(X, epsilon), options);

d = X\_opt(1);

t = X\_opt(2);

L = X\_opt(3);

% check feasibility

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

if g1 > 0 || g2 > 0 || g3 > 0 || g4 > 0 || g5 > 0 || g6 > 0 || g7 > 0 || g8 > 0 || g9 > 0 || g10 > 0

% constraints violated

else

j = j+1;

% params

e(j,:) = epsilon;

% state

X(j,:) = X\_opt;

% Force of Drag (N)

f1(j,:) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2(j,:) = eval\_f2(theta, d, L);

end

end

% update good and bad

if min(f1) < f1\_g

f1\_g = min(f1);

end

if max(f1) > f1\_b

f1\_b = max(f1);

end

if min(f2) < f2\_b

f2\_b = min(f2);

end

if max(f2) > f2\_g

f2\_g = max(f2);

end

% normalize

for i = 1:j

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

Lqinf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% plots

% normalized criterion space

figure;

plot(f1\_s, f2\_s,'.b')

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space')

% saveas(gcf,'EC\_NormalizedCriterionSpace.jpg')

% normalized criterion vs weight

figure; hold on

plot(e, f1\_s, '.')

plot(e, f2\_s, '.')

xlabel('\epsilon')

legend('f1 - drag', 'f2 - volume')

title('Normalized Criterion Space vs Epsilon')

% saveas(gcf,'EC\_NormalizedCriterionSpace\_WeightedTrend.jpg')

%% save results

save('EC\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'e', 'Lq1', 'Lq2', 'Lqinf')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function - single objective f1

function f1\_s = myfun(X)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[~, rho, mu, ~, ~, ~, ~, v, ~, ~, theta, alpha, ~, l, w] = set\_Params();

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

% Force of Drag (N)

f1 = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% scaled values

f1\_s = (f1 - f1\_g)/(f1\_b - f1\_g);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% nonlinear constraint function - constraining f2

function [c, ceq] = mycon(X, epsilon)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, ~, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[~, ~, ~, ~, ~, ~, W\_lim, FS] = set\_Lims();

% g7 = upper bound weight constraint

% g8 = lower bound buoyancy constraint

% g9 = upper bound buoyancy constraint

% g10 = upper bound stress constraint

[g7, g8, g9, g10] = eval\_g710(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w,...

d, t, L, ...

W\_lim, FS);

% good and bad values

f2\_b = 0.0353;

f2\_g = 0.1242;

% lower limit internal volume constraint

g11 = eval\_g11(d, L, f2\_g, f2\_b, epsilon);

% inequality constraints

c = [g7, g8, g9, g10, g11];

% equality constraints

ceq = [];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%% script "EpsilonConstrainedMethod\_OptVolume"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% A custom-made epsilon constrained method for multi-objective

% optimization by converting one of the two objectives into an

% inequality constraint.

% (utilizing fmincon for constrained single-objective optimization)

% There are two separate files for this method.

% This one is for optimizing volume while constraining drag.

%

%% Instructions:

% Just hit "Run". A plot of the objective functions vs epsilon is generated.

% And the results are saved in 'EC2\_results.mat'.

% Currently, saving of the plots are commented out.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% limits

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

% variable bounds

LB = [d\_L, t\_L, L\_L]; % Variable Lower Bounds

UB = [d\_U, t\_U, L\_U]; % Variable Upper Bounds

% initial guess

d0 = d\_L + (d\_U - d\_L)\*(1);

t0 = t\_L + (t\_U - t\_L)\*(1);

L0 = L\_L + (L\_U - L\_L)\*(1);

X0 = [d0, t0, L0];

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

f2\_b = 0.0353;

f2\_g = 0.1242;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% optimization / minimization

% currently these options are pretty much default, but I put this here for

% potential uses in the future

options = optimoptions(@fmincon, ...

'ConstraintTolerance', 1e-6, ...

'MaxFunctionEvaluations', 2000, ...

'MaxIterations', 1500, ...

'StepTolerance', 1e-10);

j = 0;

for i = 1:100

% multi-objective weights

epsilon = i/100;

% optimize f1, while constraining f2 <= V\_i\_lim (set in set\_Lims)

[X\_opt, f] = fmincon(@(X)myfun(X), X0, [], [], [], [], LB, UB, @(X)mycon(X, epsilon), options);

d = X\_opt(1);

t = X\_opt(2);

L = X\_opt(3);

% check feasibility

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w, ...

d, t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

if g1 > 0 || g2 > 0 || g3 > 0 || g4 > 0 || g5 > 0 || g6 > 0 || g7 > 0 || g8 > 0 || g9 > 0 || g10 > 0

% constraints violated

else

j = j+1;

% params

e(j,:) = epsilon;

% state

X(j,:) = X\_opt;

% Force of Drag (N)

f1(j,:) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L);

% Internal Volume (m^3)

f2(j,:) = eval\_f2(theta, d, L);

end

end

% update good and bad

if min(f1) < f1\_g

f1\_g = min(f1);

end

if max(f1) > f1\_b

f1\_b = max(f1);

end

if min(f2) < f2\_b

f2\_b = min(f2);

end

if max(f2) > f2\_g

f2\_g = max(f2);

end

% normalize

for i = 1:j

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

Lqinf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% plots

% normalized criterion space

figure;

plot(f1\_s, f2\_s,'.b')

xlabel('f1 - drag')

ylabel('f2 - volume')

title('Normalized Criterion Space')

axis([0,1,0,1])

% saveas(gcf,'EC2\_NormalizedCriterionSpace.jpg')

% normalized criterion vs weight

figure; hold on

plot(e, f1\_s, '.')

plot(e, f2\_s, '.')

xlabel('\epsilon')

legend('f1 - drag', 'f2 - volume')

title('Normalized Criterion Space vs Epsilon')

axis([0,1,0,1])

% saveas(gcf,'EC2\_NormalizedCriterionSpace\_WeightedTrend.jpg')

%% save results

save('EC2\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'e', 'Lq1', 'Lq2', 'Lqinf')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% objective function - single objective f2

function f2\_s = myfun(X)

% variables

d = X(1); % inner diameter of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[~, ~, ~, ~, ~, ~, ~, ~, ~, ~, theta, ~, ~, ~, ~] = set\_Params();

% good and bad values

f2\_b = 0.0353;

f2\_g = 0.1242;

% Internal Volume (m^3)

f2 = eval\_f2(theta, d, L);

% scaled values

f2\_s = (f2 - f2\_g)/(f2\_b - f2\_g);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% nonlinear constraint function - constraining f1

function [c, ceq] = mycon(X, epsilon)

% variables

d = X(1); % inner diameter of the hull (m)

t = X(2); % thickness of the hull (m)

L = X(3); % length of the cylindrical section of the hull (m)

% parameters

[g, rho, mu, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, ~, theta, alpha, tfins, l, w] = set\_Params();

% limits

[~, ~, ~, ~, ~, ~, W\_lim, FS] = set\_Lims();

% g7 = upper bound weight constraint

% g8 = lower bound buoyancy constraint

% g9 = upper bound buoyancy constraint

% g10 = upper bound stress constraint

[g7, g8, g9, g10] = eval\_g710(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, theta, alpha, tfins, l, w,...

d, t, L, ...

W\_lim, FS);

% good and bad values

f1\_g = 23.6389;

f1\_b = 25.1932;

% upper limit drag constraint

g12 = eval\_g12(rho, mu, v, theta, alpha, l, w, d, t, L, f1\_g, f1\_b, epsilon);

% inequality constraints

c = [g7, g8, g9, g10, g12];

% equality constraints

ceq = [];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "Plot\_Opt\_on\_Feas"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Evaluates the inequality constraints for a full 3D cubic grid of

% points spanning the variable bounds. Clusters the evaluated points

% according to the inequality constraints violated. Generates a plot

% showing the various regions.

% The solid green region is the feasible domain.

% The transparent grey region violates the stress constraint.

% The transparent blue region violates the buoyancy constraints.

% The transparent red region violates the weight constraint.

% Note that these infeasible regions overlap.

% Also plots the optima saved in 'fmincon\_results.mat' on top.

%

%% Instructions:

% (Check that 'fmincon\_results.mat' has been created and is in the

% current folder...

% ...Note: fmincon could be easily replaced to plot other optima)

% Just hit "Run".

% It will generate a 3D plot of the deign space showing the feasible

% domain and infeasible regions.

% Hit any button to continue when ready.

% It will add the optima points to the plot.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% clean up

close all

clear

clc

%% Variable Definitions:

% d inner diameter of the hull (m)

% t thickness of the hull (m)

% L length of the cylindrical section of the hull (m)

%% get all parameters:

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

%% Variable Bounds:

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

%% EVAL for all 3 variables

N = 50;

% init

W = zeros(N,N,N);

F\_B = zeros(N,N,N);

s\_max = zeros(N,N,N);

F\_D = zeros(N,N,N);

V\_i = zeros(N,N,N);

W\_load = zeros(N,N,N);

F\_L = zeros(N,N,N);

W\_T = zeros(N,N,N);

d = linspace(d\_L, d\_U, N);

t = linspace(t\_L, t\_U, N);

L = linspace(L\_L, L\_U, N);

XR = [];

XB = [];

XK = [];

XG = [];

figure(1);

hold on

for i = 1:N

for j = 1:N

for k = 1:N

% Calculate Everything

[W(i,j,k), F\_B(i,j,k), s\_max(i,j,k), F\_D(i,j,k), V\_i(i,j,k), W\_load(i,j,k), F\_L(i,j,k)] = ...

calc\_Everything(d(i), t(j), L(k));

W\_T(i,j,k) = W(i,j,k) + W\_load(i,j,k);

% CHECK CONSTRAINTS

% stress - black

if s\_max(i,j,k) - (1/FS)\*Sy\_hull > 0

if isempty(XK)

XK = d(i);

YK = t(j);

ZK = L(k);

else

XK(end+1,1) = d(i);

YK(end+1,1) = t(j);

ZK(end+1,1) = L(k);

end

end

% buoyancy - blue

if F\_B(i,j,k) - W\_T(i,j,k) - F\_L(i,j,k) > 0 || W\_T(i,j,k) - F\_B(i,j,k) - F\_L(i,j,k) > 0

if isempty(XB)

XB = d(i);

YB = t(j);

ZB = L(k);

else

XB(end+1,1) = d(i);

YB(end+1,1) = t(j);

ZB(end+1,1) = L(k);

end

end

% weight - red

if W\_T(i,j,k) - W\_lim > 0

if isempty(XR)

XR = d(i);

YR = t(j);

ZR = L(k);

else

XR(end+1,1) = d(i);

YR(end+1,1) = t(j);

ZR(end+1,1) = L(k);

end

end

% everything passed - green

if s\_max(i,j,k) - (1/FS)\*Sy\_hull > 0

elseif F\_B(i,j,k) - W\_T(i,j,k) - F\_L(i,j,k) > 0 || W\_T(i,j,k) - F\_B(i,j,k) - F\_L(i,j,k) > 0

elseif W\_T(i,j,k) - W\_lim > 0

else

if isempty(XG)

XG = d(i);

YG = t(j);

ZG = L(k);

else

XG(end+1,1) = d(i);

YG(end+1,1) = t(j);

ZG(end+1,1) = L(k);

end

end

end

end

end

%% create shapes

% buoyancy - blue

if ~isempty(XB)

SB = alphaShape(XB, YB, ZB);

SB.Alpha = 0.05;

plot(SB,'FaceColor','b','FaceAlpha',0.25,'EdgeColor','none');

end

% weight - red

if ~isempty(XR)

SR = alphaShape(XR, YR, ZR);

SR.Alpha = 0.05;

plot(SR,'FaceColor','r','FaceAlpha',0.25,'EdgeColor','none');

end

% stress - black

if ~isempty(XK)

SK = alphaShape(XK, YK, ZK);

SK.Alpha = 0.05;

plot(SK,'FaceColor','k','FaceAlpha',0.25,'EdgeColor','none');

end

% everything passed - green

if ~isempty(XG)

SG = alphaShape(XG, YG, ZG);

% SG.Alpha = 0.05;

plot(SG,'FaceColor','g');

end

xlabel('d (m)')

ylabel('t (m)')

zlabel('L (m)')

title('Feasible Domain - Design Space')

axis([d\_L d\_U t\_L t\_U L\_L L\_U])

axis square

view(-45,15)

% saveas(gcf, 'FeasibleDomain\_DesignSpace.jpg')

pause();

%% load in optima

ALL\_res = load('fmincon\_results.mat'); % this can be changed

X = ALL\_res.X;

%% plot optima

plot3(X(:,1), X(:,2), X(:,3), 'ok', 'MarkerFaceColor', 'k')

title('Optima - Design Space')

% saveas(gcf, 'Optima\_on\_FeasibleDomain\_DesignSpace.jpg')

% %% make gif

% make\_animated\_gif('clear')

%

% for i = 1:360

% view(-45 + i,15)

% make\_animated\_gif('snap', gcf)

% end

% make\_animated\_gif('write','Optima\_on\_FeasibleDomain\_DesignSpace', 0.1)

%% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "CheckFeas"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Checks optima for feasibility

%

%% Instructions:

% Fill in the variable x with the optima point, then hit "Run"

% It will display the values for each of the inequality constraints.

% It will display Y/N depending on if the point is feasible.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% clean up

close all

clear

clc

%% point to be evaluated

x = [x1, x2, x3]; % to be filled in by user\*\*\*

d = x(1);

t = x(2);

L = x(3);

%% get all parameters:

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, T, theta, alpha, tfins, l, w] = set\_Params();

%% Variable Bounds:

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

%% check feasibility

[g1, g2, g3, g4, g5, g6, g7, g8, g9, g10] = ...

eval\_gALL(g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, theta, alpha, tfins, l, w, d, t, L, d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS)

if g1 > 0 || g2 > 0 || g3 > 0 || g4 > 0 || g5 > 0 || g6 > 0 || g7 > 0 || g8 > 0 || g9 > 0 || g10 > 0

fprintf('\n\n N \n\n')

else

fprintf('\n\n Y \n\n')

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "Compile\_All\_Opt"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Combines all the optima generated by the various methods into a

% single set which is saved as 'ALL\_results.mat'.

%

%% Instructions:

% Verify that all the following files exist before hitting 'Run':

% 'fmincon\_results.mat'

% 'penalty\_results.mat'

% 'penalty2\_results.mat'

% ('ALM\_results.mat')

% 'fminimax\_results.mat'

% 'Lq1\_results.mat'

% 'Lq2\_results.mat'

% 'LqInf\_results.mat'

% 'EC\_results.mat'

% 'EC2\_results.mat'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% clean up

close all

clear

clc

%% fmincon

fmincon\_res = load('fmincon\_results.mat');

X\_fmincon = fmincon\_res.X;

f1\_fmincon = fmincon\_res.f1;

f2\_fmincon = fmincon\_res.f2;

%% Penalty Method

penalty\_res = load('penalty\_results.mat');

X\_penalty = penalty\_res.X;

f1\_penalty = penalty\_res.f1;

f2\_penalty = penalty\_res.f2;

penalty2\_res = load('penalty2\_results.mat');

X\_penalty2 = penalty2\_res.X;

f1\_penalty2 = penalty2\_res.f1;

f2\_penalty2 = penalty2\_res.f2;

X\_penalty = [X\_penalty; X\_penalty2];

f1\_penalty = [f1\_penalty; f1\_penalty2];

f2\_penalty = [f2\_penalty; f2\_penalty2];

%% Augmented Lagrangian Method (ALM)

ALM\_res = load('ALM\_results.mat');

X\_ALM = ALM\_res.X;

f1\_ALM = ALM\_res.f1;

f2\_ALM = ALM\_res.f2;

%% fminimax

fminimax\_res = load('fminimax\_results.mat');

X\_fminimax = fminimax\_res.X\_opt;

f1\_fminimax = fminimax\_res.f1;

f2\_fminimax = fminimax\_res.f2;

%% Global Criterion (Lq)

Lq1\_res = load('Lq1\_results.mat');

X\_Lq1 = Lq1\_res.X\_opt;

f1\_Lq1 = Lq1\_res.f1;

f2\_Lq1 = Lq1\_res.f2;

Lq2\_res = load('Lq2\_results.mat');

X\_Lq2 = Lq2\_res.X\_opt;

f1\_Lq2 = Lq2\_res.f1;

f2\_Lq2 = Lq2\_res.f2;

LqInf\_res = load('LqInf\_results.mat');

X\_LqInf = LqInf\_res.X\_opt;

f1\_LqInf = LqInf\_res.f1;

f2\_LqInf = LqInf\_res.f2;

X\_Lq = [X\_Lq1; X\_Lq2; X\_LqInf];

f1\_Lq = [f1\_Lq1; f1\_Lq2; f1\_LqInf];

f2\_Lq = [f2\_Lq1; f2\_Lq2; f2\_LqInf];

%% Epsilon Constrained (EC)

EC\_res = load('EC\_results.mat');

X\_EC = EC\_res.X;

f1\_EC = EC\_res.f1;

f2\_EC = EC\_res.f2;

EC2\_res = load('EC2\_results.mat');

X\_EC2 = EC2\_res.X;

f1\_EC2 = EC2\_res.f1;

f2\_EC2 = EC2\_res.f2;

X\_EC = [X\_EC; X\_EC2];

f1\_EC = [f1\_EC; f1\_EC2];

f2\_EC = [f2\_EC; f2\_EC2];

%% compile ALL

X = [X\_fmincon; X\_penalty; X\_fminimax; X\_Lq; X\_EC];

f1 = [f1\_fmincon; f1\_penalty; f1\_fminimax; f1\_Lq; f1\_EC];

f2 = [f2\_fmincon; f2\_penalty; f2\_fminimax; f2\_Lq; f2\_EC];

f1\_g = min(f1);

f1\_b = max(f1);

f2\_b = min(f2);

f2\_g = max(f2);

% normalize

for i = 1:max(size(f1))

% scaled values

f1\_s(i,:) = (f1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s(i,:) = (f2(i) - f2\_g)/(f2\_b - f2\_g);

% multi-objective Lq-method

Lq1(i,:) = sum([f1\_s(i), f2\_s(i)]);

Lq2(i,:) = sqrt(sumsqr([f1\_s(i), f2\_s(i)]));

LqInf(i,:) = max([f1\_s(i), f2\_s(i)]);

end

%% save

save('ALL\_results.mat', 'X', 'f1', 'f2', 'f1\_s', 'f2\_s', 'Lq1', 'Lq2', 'LqInf');

%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "Plot\_All\_Opt\_wColors"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Plots all the optima generated by the various methods as

% different colors/markers in the normalized criterion space.

%

%% Instructions:

% Verify that all the following files exist before hitting 'Run':

% 'GRID\_results.mat'

% 'ALL\_results.mat'

% 'fmincon\_results.mat'

% 'penalty\_results.mat'

% 'penalty2\_results.mat'

% ('ALM\_results.mat')

% 'fminimax\_results.mat'

% 'Lq1\_results.mat'

% 'Lq2\_results.mat'

% 'LqInf\_results.mat'

% 'EC\_results.mat'

% 'EC2\_results.mat'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% clean up

close all

clear

clc

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, ~, ~] = set\_Lims();

%% GRID

GRID\_res = load('GRID\_results.mat');

f1\_s\_GRID = GRID\_res.f1\_s;

f2\_s\_GRID = GRID\_res.f2\_s;

%% all "optima"

ALL\_res = load('ALL\_results.mat');

X = ALL\_res.X;

f1 = ALL\_res.f1;

f2 = ALL\_res.f2;

f1\_s = ALL\_res.f1\_s;

f2\_s = ALL\_res.f2\_s;

Lq1 = ALL\_res.Lq1;

Lq2 = ALL\_res.Lq2;

LqInf = ALL\_res.LqInf;

% select the Lq1 norm from the full set of all optima

[~, idx] = min(Lq1);

X\_LQ1 = X(idx, :);

f1\_LQ1 = f1(idx);

f2\_LQ1 = f2(idx);

f1\_s\_LQ1 = f1\_s(idx);

f2\_s\_LQ1 = f2\_s(idx);

% select the Lq2 norm from the full set of all optima

[~, idx] = min(Lq2);

X\_LQ2 = X(idx, :);

f1\_LQ2 = f1(idx);

f2\_LQ2 = f2(idx);

f1\_s\_LQ2 = f1\_s(idx);

f2\_s\_LQ2 = f2\_s(idx);

% select the LqInf norm from the full set of all optima

[~, idx] = min(LqInf);

X\_LQinf = X(idx, :);

f1\_LQinf = f1(idx);

f2\_LQinf = f2(idx);

f1\_s\_LQinf = f1\_s(idx);

f2\_s\_LQinf = f2\_s(idx);

% extract good and bad values from all optima

f1\_g = min(f1);

f1\_b = max(f1);

f2\_b = min(f2);

f2\_g = max(f2);

%% fmincon

fmincon\_res = load('fmincon\_results.mat');

X\_fmincon = fmincon\_res.X;

f1\_fmincon = fmincon\_res.f1;

f2\_fmincon = fmincon\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_fmincon))

f1\_s\_fmincon(i,:) = (f1\_fmincon(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_fmincon(i,:) = (f2\_fmincon(i) - f2\_g)/(f2\_b - f2\_g);

end

%% Penalty Method

penalty\_res = load('penalty\_results.mat');

X\_penalty = penalty\_res.X;

f1\_penalty = penalty\_res.f1;

f2\_penalty = penalty\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_penalty))

f1\_s\_penalty(i,:) = (f1\_penalty(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_penalty(i,:) = (f2\_penalty(i) - f2\_g)/(f2\_b - f2\_g);

end

% second chosen start point

penalty2\_res = load('penalty2\_results.mat');

X\_penalty2 = penalty2\_res.X;

f1\_penalty2 = penalty2\_res.f1;

f2\_penalty2 = penalty2\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_penalty2))

f1\_s\_penalty2(i,:) = (f1\_penalty2(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_penalty2(i,:) = (f2\_penalty2(i) - f2\_g)/(f2\_b - f2\_g);

end

% combine both sets of the penalty method results

X\_penalty = [X\_penalty; X\_penalty2];

f1\_penalty = [f1\_penalty; f1\_penalty2];

f2\_penalty = [f2\_penalty; f2\_penalty2];

f1\_s\_penalty = [f1\_s\_penalty; f1\_s\_penalty2];

f2\_s\_penalty = [f2\_s\_penalty; f2\_s\_penalty2];

% %% Augmented Lagrangian Method (ALM)

% ALM\_res = load('ALM\_results.mat');

% X\_ALM = ALM\_res.X;

% f1\_ALM = ALM\_res.f1;

% f2\_ALM = ALM\_res.f2;

%

%

% for i = 1:max(size(f1\_ALM))

% f1\_s\_ALM(i,:) = (f1\_ALM(i) - f1\_g)/(f1\_b - f1\_g);

% f2\_s\_ALM(i,:) = (f2\_ALM(i) - f2\_g)/(f2\_b - f2\_g);

% end

%% fminimax

fminimax\_res = load('fminimax\_results.mat');

X\_fminimax = fminimax\_res.X\_opt;

f1\_fminimax = fminimax\_res.f1;

f2\_fminimax = fminimax\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_fminimax))

f1\_s\_fminimax(i,:) = (f1\_fminimax(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_fminimax(i,:) = (f2\_fminimax(i) - f2\_g)/(f2\_b - f2\_g);

end

%% Global Criterion (Lq)

Lq1\_res = load('Lq1\_results.mat');

X\_Lq1 = Lq1\_res.X\_opt;

f1\_Lq1 = Lq1\_res.f1;

f2\_Lq1 = Lq1\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_Lq1))

f1\_s\_Lq1(i,:) = (f1\_Lq1(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_Lq1(i,:) = (f2\_Lq1(i) - f2\_g)/(f2\_b - f2\_g);

end

Lq2\_res = load('Lq2\_results.mat');

X\_Lq2 = Lq2\_res.X\_opt;

f1\_Lq2 = Lq2\_res.f1;

f2\_Lq2 = Lq2\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_Lq2))

f1\_s\_Lq2(i,:) = (f1\_Lq2(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_Lq2(i,:) = (f2\_Lq2(i) - f2\_g)/(f2\_b - f2\_g);

end

LqInf\_res = load('LqInf\_results.mat');

X\_LqInf = LqInf\_res.X\_opt;

f1\_LqInf = LqInf\_res.f1;

f2\_LqInf = LqInf\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_LqInf))

f1\_s\_LqInf(i,:) = (f1\_LqInf(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_LqInf(i,:) = (f2\_LqInf(i) - f2\_g)/(f2\_b - f2\_g);

end

% combine all results obtained by the Global Criterion method

X\_Lq = [X\_Lq1; X\_Lq2; X\_LqInf];

f1\_Lq = [f1\_Lq1; f1\_Lq2; f1\_LqInf];

f2\_Lq = [f2\_Lq1; f2\_Lq2; f2\_LqInf];

f1\_s\_Lq = [f1\_s\_Lq1; f1\_s\_Lq2; f1\_s\_LqInf];

f2\_s\_Lq = [f2\_s\_Lq1; f2\_s\_Lq2; f2\_s\_LqInf];

%% Epsilon Constrained (EC)

% optimizing for drag while constraining volume

EC\_res = load('EC\_results.mat');

X\_EC = EC\_res.X;

f1\_EC = EC\_res.f1;

f2\_EC = EC\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_EC))

f1\_s\_EC(i,:) = (f1\_EC(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_EC(i,:) = (f2\_EC(i) - f2\_g)/(f2\_b - f2\_g);

end

% optimizing for volume, while constraining drag

EC2\_res = load('EC2\_results.mat');

X\_EC2 = EC2\_res.X;

f1\_EC2 = EC2\_res.f1;

f2\_EC2 = EC2\_res.f2;

% re-normalize according to full set of optima

for i = 1:max(size(f1\_EC2))

f1\_s\_EC2(i,:) = (f1\_EC2(i) - f1\_g)/(f1\_b - f1\_g);

f2\_s\_EC2(i,:) = (f2\_EC2(i) - f2\_g)/(f2\_b - f2\_g);

end

% combine both sets of optima obtained by the epsilon constrained method

X\_EC = [X\_EC; X\_EC2];

f1\_EC = [f1\_EC; f1\_EC2];

f2\_EC = [f2\_EC; f2\_EC2];

f1\_s\_EC = [f1\_s\_EC; f1\_s\_EC2];

f2\_s\_EC = [f2\_s\_EC; f2\_s\_EC2];

%% plot according to color and marker type

figure; hold on

plot(f1\_s\_GRID, f2\_s\_GRID, '.', 'color', [0.5, 0.5, 0.5])

plot(f1\_s\_EC, f2\_s\_EC, '.g')

plot(f1\_s\_penalty, f2\_s\_penalty, '.r')

plot(f1\_s\_fmincon, f2\_s\_fmincon, '.c')

% plot(f1\_s\_ALM, f2\_s\_ALM, '.c')

plot(f1\_s\_fminimax, f2\_s\_fminimax, 'ob', 'MarkerFaceColor','b')

plot(f1\_s\_Lq, f2\_s\_Lq, 'ok')

plot(f1\_s\_LQ1, f2\_s\_LQ1, '\*m')

plot(f1\_s\_LQ2, f2\_s\_LQ2, '\*m')

plot(f1\_s\_LQinf, f2\_s\_LQinf, '\*m')

legend('Grid','Epsilon Constrained', 'Penalty', 'fmincon', 'fminimax', 'Global Criterion', 'Gobal Criterion (2)')

title('Normalized Criterion Space')

xlabel('f1 - drag')

ylabel('f2 - volume')

axis([0, 1, 0, 1]);

axis on

% saveas(gcf,'AllOpt\_wColor\_NormalizedCriterionSpace.jpg')

%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "isolate\_Pareto"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Isolates the Pareto Frontier from a set of points

%% Instructions:

% (Check that 'GRID\_results.mat' and 'ALL\_results.mat' have been

% created and were saved in the current folder.)

% Just hit "Run".

% It will plot all the points in the feasible domain in grey.

% It will plot the Pareto Frontier in green.

% It will save the plot as 'Pareto.jpg'

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% clean up

close all

clear

clc

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, ~, ~] = set\_Lims();

%% Grid Eval

GRID\_res = load('GRID\_results.mat');

f1\_s = GRID\_res.f1\_s;

f2\_s = GRID\_res.f2\_s;

%% plot grid

% normalized criterion space

figure(3); hold on

plot(f1\_s, f2\_s,'.', 'color', [0.5, 0.5, 0.5])

xlabel('f1')

ylabel('f2')

title('normalized criterion space')

%% All Optima

ALL\_res = load('ALL\_results.mat');

f1\_s = ALL\_res.f1\_s;

f2\_s = ALL\_res.f2\_s;

%% find the pareto points

k = 0;

for i = max(size(f1\_s)):-1:1

np = 0;

for j = 1:i-1 % check all other values

if f1\_s(i) > f1\_s(j) && f2\_s(i) > f1\_s(j) % if dominated by another point in terms of both objective functions

% not pareto

np = 1;

end

end

if ~np

% pareto

k = k+1;

f1\_s\_p(k,:) = f1\_s(i,:);

f2\_s\_p(k,:) = f2\_s(i,:);

end

end

% update

f1\_s = f1\_s\_p;

f2\_s = f2\_s\_p;

f1\_s\_p = [];

f2\_s\_p = [];

k = 0;

for i = max(size(f1\_s)):-1:1

np = 0;

for j = 1:i-1 % check all other values

if f1\_s(i) == f1\_s(j) && f2\_s(i) > f1\_s(j) || f2\_s(i) == f2\_s(j) && f1\_s(i) > f1\_s(j) % if dominated by another point in terms of just one of the objectives

% not pareto

np = 1;

end

end

if ~np

% pareto

k = k+1;

f1\_s\_p(k,:) = f1\_s(i,:);

f2\_s\_p(k,:) = f2\_s(i,:);

end

end

%% plot Pareto points

plot(f1\_s\_p, f2\_s\_p,'.g')

%% save

saveas(gcf, 'Pareto.jpg')

%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "Monotonicity\_Analysis\_d"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Plots the objective functions and inequality constraints as functions

% of a single variable: d

%

%% Instructions:

% Just hit "Run". A plot will be generated.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% parameters

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, T, theta,alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, ...

W\_lim, FS] = set\_Lims();

%% test for a single variable at a time by holding the others constant:

% constant

t = (t\_L + t\_U)/2;

L = (L\_L + L\_U)/2;

% varying

N = 100;

d = linspace(d\_L, d\_U, N);

%% test

for i = 1:N

% Force of Drag (N)

f1(i) = eval\_f1(rho, mu, v, theta, alpha, l, w, d(i), t, L);

% Internal Volume (m^3)

f2(i) = eval\_f2(theta, d(i), L);

% constraints

[g1(i), g2(i), g3(i), g4(i), g5(i), g6(i), g7(i), g8(i), g9(i), g10(i)] = ...

eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, theta, alpha, tfins, l, w,...

d(i), t, L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

end

% rescale / normalize outputs from +/- 0 to 1

f1 = f1/max(abs(f1));

f2 = f2/max(abs(f2));

g1 = g1/max(abs(g1));

g2 = g2/max(abs(g2));

g7 = g7/max(abs(g7));

g8 = g8/max(abs(g8));

g9 = g9/max(abs(g9));

g10 = g10/max(abs(g10));

% plot

figure(1);

hold on

plot(d, f1, 'g', 'LineWidth', 2);

plot(d, f2, 'm', 'LineWidth', 2);

plot(d, g1, 'y', 'LineWidth', 2);

plot(d, g2, 'c', 'LineWidth', 2);

plot(d, g7, 'r', 'LineWidth', 2);

plot(d, g8, 'b', 'LineWidth', 2);

plot(d, g9, 'b', 'LineWidth', 2);

plot(d, g10, 'k', 'LineWidth', 2);

title('Monotonic ?')

xlabel('d (m)');

ylabel('f & g');

legend({'f1','f2','g1','g2','g7','g8','g9','g10'}, 'Location', 'EastOutside');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "Monotonicity\_Analysis\_t"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Plots the objective functions and inequality constraints as functions

% of a single variable: t

%

%% Instructions:

% Just hit "Run". A plot will be generated.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% parameters

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, T, theta,alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

%% test for a single variable at a time by holding the others constant:

% constant

d = (d\_L + d\_U)/2;

L = (L\_L + L\_U)/2;

% varying

N = 100;

t = linspace(t\_L, t\_U, N);

%% test

for i = 1:N

% Force of Drag (N)

f1(i) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t(i), L);

% Internal Volume (m^3)

f2(i) = eval\_f2(theta, d, L);

% constraints

[g1(i), g2(i), g3(i), g4(i), g5(i), g6(i), g7(i), g8(i), g9(i), g10(i)] = ...

eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, theta, alpha, tfins, l, w,...

d, t(i), L, ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

end

% rescale / normalize outputs from +/- 0 to 1

f1 = f1/max(abs(f1));

f2 = f2/max(abs(f2));

g3 = g3/max(abs(g3));

g4 = g4/max(abs(g4));

g7 = g7/max(abs(g7));

g8 = g8/max(abs(g8));

g9 = g9/max(abs(g9));

g10 = g10/max(abs(g10));

% plot

figure(1);

hold on

plot(t, f1, 'g', 'LineWidth', 2);

plot(t, f2, 'm', 'LineWidth', 2);

plot(t, g3, 'y', 'LineWidth', 2);

plot(t, g4, 'c', 'LineWidth', 2);

plot(t, g7, 'r', 'LineWidth', 2);

plot(t, g8, 'b', 'LineWidth', 2);

plot(t, g9, 'b', 'LineWidth', 2);

plot(t, g10, 'k', 'LineWidth', 2);

title('Monotonic ?')

xlabel('t (m)');

ylabel('f & g');

legend({'f1','f2','g3','g4','g7','g8','g9','g10'}, 'Location', 'EastOutside');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% script "Monotonicity\_Analysis\_L"

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Plots the objective functions and inequality constraints as functions

% of a single variable: L

%

%% Instructions:

% Just hit "Run". A plot will be generated.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set up

close all

clear

clc

% parameters

[g, rho, mu, ...

rho\_load, rho\_fins, rho\_hull, Sy\_hull, ...

v, depth, T, theta,alpha, tfins, l, w] = set\_Params();

% limits

[d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS] = set\_Lims();

%% test for a single variable at a time by holding the others constant:

% constant

d = (d\_L + d\_U)/2;

t = (t\_L + t\_U)/2;

% varying

N = 100;

L = linspace(L\_L, L\_U, N);

%% test

for i = 1:N

% Force of Drag (N)

f1(i) = eval\_f1(rho, mu, v, theta, alpha, l, w, d, t, L(i));

% Internal Volume (m^3)

f2(i) = eval\_f2(theta, d, L(i));

% constraints

[g1(i), g2(i), g3(i), g4(i), g5(i), g6(i), g7(i), g8(i), g9(i), g10(i)] = ...

eval\_gALL(...

g, rho, rho\_load, rho\_fins, rho\_hull, Sy\_hull, v, depth, theta, alpha, tfins, l, w,...

d, t, L(i), ...

d\_L, d\_U, t\_L, t\_U, L\_L, L\_U, W\_lim, FS);

end

% rescale / normalize outputs from +/- 0 to 1

f1 = f1/max(abs(f1));

f2 = f2/max(abs(f2));

g5 = g5/max(abs(g5));

g6 = g6/max(abs(g6));

g7 = g7/max(abs(g7));

g8 = g8/max(abs(g8));

g9 = g9/max(abs(g9));

g10 = g10/max(abs(g10));

% plot

figure(1);

hold on

plot(L, f1, 'g', 'LineWidth', 2);

plot(L, f2, 'm', 'LineWidth', 2);

plot(L, g5, 'y', 'LineWidth', 2);

plot(L, g6, 'c', 'LineWidth', 2);

plot(L, g7, 'r', 'LineWidth', 2);

plot(L, g8, 'b', 'LineWidth', 2);

plot(L, g9, 'b', 'LineWidth', 2);

plot(L, g10, 'k', 'LineWidth', 2);

title('Monotonic ?')

xlabel('L (m)');

ylabel('f & g');

legend({'f1','f2','g5','g6','g7','g8','g9','g10'}, 'Location', 'EastOutside');

function make\_animated\_gif(varargin)

%% Description:

% function to build an animated gif

%% Instructions:

% There a few different inputs needed

% at various points within a function to successfully create a gif.

%

% First, use make\_animated\_gif('clear') to start a new gif.

%

% Then for each frame of the gif, use make\_animated\_gif('snap', [handle]),

%

% and when done with all the frames,

% use make\_animated\_gif('write', filename, [delayTime], [loopCount])

% to generate the gif file.

%% Inputs:

% filename - name of the file you want to make

% delayTime - Delay time between images (s, default is 0)

% loopCount - Number of times to loop gif (default is infinite)

%% Outputs:

% [filename].gif - the animated gif file

%%

global ANIMATED\_GIF\_IM ANIMATED\_GIF\_MAP ANIMATED\_GIF\_PREV

command = varargin{1};

if strcmp(command,'clear')

% ======================================================================= %

% ======================================================================= %

ANIMATED\_GIF\_IM=[];

ANIMATED\_GIF\_MAP=[];

% ======================================================================= %

% ======================================================================= %

elseif strcmp(command,'snap')

% ======================================================================= %

% ======================================================================= %

if nargin>=2

handle=varargin{2};

else

handle=gcf;

end

set(gcf,'Renderer','zbuffer')

if isempty(ANIMATED\_GIF\_IM)

drawnow

f = getframe(handle);

[ANIMATED\_GIF\_IM,ANIMATED\_GIF\_MAP] = ...

rgb2ind(f.cdata,256,'nodither');

else

f = getframe(handle);

try

[new\_im,new\_map] = rgb2ind(f.cdata,256,'nodither'); % Account for any new colors to be added to the map

ANIMATED\_GIF\_MAP = [ANIMATED\_GIF\_MAP;

setdiff(new\_map,ANIMATED\_GIF\_MAP,'rows')];

ANIMATED\_GIF\_IM(:,:,1,end+1) = ...

rgb2ind(f.cdata,ANIMATED\_GIF\_MAP,'nodither');

catch

disp('Skipping differently sized image')

end

end

% ======================================================================= %

% ======================================================================= %

elseif strcmp(command,'write')

% ======================================================================= %

% ======================================================================= %

filename=varargin{2};

if nargin>=3

delayTime=varargin{3};

end

if nargin>=4

loopCount=varargin{4};

end

[pathstr name ext]=fileparts(filename);

if ~isdir(pathstr) && ~isempty(pathstr)

error('Path does not exist: %s\n',pathstr);

end

disp(['Writing to: ' fullfile(pathstr,[name '.gif'])])

if isempty(ANIMATED\_GIF\_IM) && isfield(ANIMATED\_GIF\_PREV,'IM')

ANIMATED\_GIF\_IM = ANIMATED\_GIF\_PREV.IM;

ANIMATED\_GIF\_MAP= ANIMATED\_GIF\_PREV.MAP;

end

im=ANIMATED\_GIF\_IM;

map=ANIMATED\_GIF\_MAP;

if ~exist('delayTime')

delayTime=0;

end

if ~exist('loopCount')

loopCount=inf;

end

if isempty(im)

error('Nothing to write.')

else

imwrite(im,map,fullfile(pathstr,[name '.gif']),...

'DelayTime',delayTime,'LoopCount',loopCount);

end

ANIMATED\_GIF\_PREV.IM = ANIMATED\_GIF\_IM;

ANIMATED\_GIF\_PREV.MAP= ANIMATED\_GIF\_MAP;

ANIMATED\_GIF\_IM=[];

ANIMATED\_GIF\_MAP=[];

% ======================================================================= %

% ======================================================================= %

else

% ======================================================================= %

% ======================================================================= %

error('Command not understood: %s',command)

% ======================================================================= %

% ======================================================================= %

end

end

function [v] = calc\_vmax(g, rho, mu, D, L\_T, V, S, A, T, C\_D\_fins, W)

% ENME 610 - Engineering Optimization

% University of Maryland, College Park

% Group 1: David Smart, Luke Travisiano, Jason Morin

% AUV Optimization

%

%% Description:

% Calculates the Terminal Velocity of AUV,

% This function was created just to check that our calculations were

% correct, and confirmed them. It showed that the AUV could reach or

% exceed the set parameter value for velocity (v) when the drag

% calculated at v was less than the available thrust T.

% This function could potentially be used as another objective in a

% future iteration of this project.

%

% Also, currently set up to make a plot for drag force, net forward force,

% and speed vs time.

%

%% Inputs:

% g acceleration due to gravity (m/s^2)

% rho density of water at 25 deg-C (kg/m^3)

% mu dynamic viscosity of water at 25 deg-C (N\*s/m^2)

% D outer diameter of the hull (m)

% L\_T total length of the hull (m)

% V volume of water displaced by the hull (m^3)

% S surface area of the hull (m^2)

% A cross-section area of fins (m^2)

% T thrust from the AUV's propeller (N)

% C\_D\_fins Drag Coefficient for fins (n/a - unitless)

% W weight of the empty AUV (N)

%

%% Outputs:

% v terminal velocity (m/s)

%

a = 1; % just to get into the loop

% initialize

F\_D = 0; v = 0;

t = 0; dt = 0.01;

% stopping criteria

tol = 1\*10^-6;

figure; hold on

while a > tol

% calc thrust - drag

F = T - F\_D;

% calc acceleration

a = F/(W/g);

% calc velocity

v = v + a\*dt;

% Reynold's number

Rn = calc\_Rn(rho, mu, v, L\_T);

% Drag Coefficient for hull (n/a - unitless)

[C\_v] = calc\_HydroCoeff(D, L\_T, V, Rn);

% calc drag

F\_D\_hull = C\_v\*(rho/2)\*S\*v^2;

F\_D\_fin = C\_D\_fins\*(rho/2)\*A\*v^2;

F\_D = F\_D\_hull + F\_D\_fin;

t = t+dt;

% plot

plot(t, F\_D,'r.')

plot(t, F,'g.')

plot(t, v,'b.')

end

pause(0.1)

end