Memo

Aero_M_Optimization_Algorithms_10-07-31.doc

Date: 2010-07-31

From:
Joseph Yang
Rice University
6100 Main St.
Houston, TX 77005
The United States of America

E-mail: j.yang@rice.edu

To:
Mihaela Niță (Mihaela.Nita@haw-hamburg.de)
HAW Hamburg

Copy to:
Dieter Scholz (info@Prof.Scholz.de)
HAW Hamburg

Optimization Algorithms for Aircraft Preliminary Sizing and Cabin Design

1 Introduction

1.1. Motivation and Scope of Work

As new aircraft are being designed, optimization of the design parameters becomes necessary to decrease fuel costs and emissions and maximize profits. To do this, after requirements are decided on, such as aircraft configuration, design range or payload, design parameters can be optimized to achieve the most efficient aircraft. As opposed to trial-and-error optimization (**Kroo 2006**) where a design may go through several rounds of testing to determine efficiency, optimization algorithms can save time and effort when implemented properly.

Algorithms can be applied to either optimize aircraft performance (the view of aircraft design) or aircraft comfort (the view of aircraft cabin design as part of aircraft design).

This Memo is part of the research towards a Ph.D with the topic: '*Contributions to the Optimization Methodology for Aircraft Cabin Design and Conversion*'. The aim is to program a set of algorithms applicable to aircraft preliminary sizing. Cabin design cannot be performed without the main parameters from the aircraft preliminary sizing.

The algorithms will be tested using a simple eight variable sample function and a four variable function representing the 'drag in the responsibility of the cabin'.

1.2 Optimization Approach

Currently aircraft preliminary sizing is optimized using EXCEL (**Raymer 2002**). At HAW Hamburg a tool called PreSTo (Preliminary Sizing Tool) developed in EXCEL delivering the main parameters of aircraft preliminary sizing is under development.

The approach used here was:
- Selecting applicable algorithm from two main classes: stochastic and deterministic,
- Coding the algorithms in MATLAB,
- Testing the algorithms with a sample function,
- Concluding upon the efficiency of each algorithm in connection to the optimization problem.

The algorithms delivered by this Memo are to be used in the future work for optimizing several objective functions, such as:
- Minimum take-off mass
- Minimum operating empty mass
- Minimum thrust
- Minimum fuel
- Minimum reference area
- Minimum DOC (Direct Operating Costs)

The results of the optimization algorithms can input into PreSTo for a double check.

By default, the optimization algorithms are coded with a constraint of ± 20 % from the initial guess. Lower and upper boundaries can be as well defined.

The sample function is an eight design variables function. These variables are:
- Thrust-to-weight ratio
- Ratio of weight to wing area
- Aspect Ratio
- Wing Sweep
- Taper
- Thickness
- Fineness ratio, or Slenderness (sometimes denoted by the Greek character λ)
- Design lift coefficient

1.3 Optimization Algorithms

Optimization algorithms are of two types: stochastic and deterministic. Raymer is an author that already tested part of these algorithms, but not for preliminary sizing but for the aircraft design as a whole (**Raymer 2002**).

In a deterministic scheme, given the possible inputs, the simulation always returns the same result. There is no chance involved or reliance on a probability distribution in the algorithm.

Stochastic algorithms rely on some kind of probability distribution or random chance to

predict a likely best answer. For instance, the stochastic algorithms examined were coded using MATLAB's "rand" as an element of random chance, which dictates the way the system progresses towards a solution. As such, stochastic methods benefit from several random starts and may end up in different local optima, which allows the design to be more flexible.

The stochastic methods used were:
- Random Monte Carlo,
- Gaussian Random Walk,
- Simulated Annealing.

The deterministic method examined was the method of Orthogonal Steepest Descent.

2 Description and Analysis of Optimization Algorithms

2.1 Random Monte Carlo

The way chosen to implement this algorithm is to sample the search space repeatedly and at random using "rand," MATLAB's uniform probability generator. This is accomplished by providing a starting guess, and then the program will establish a ±20% boundary from which it samples values randomly (**Raymer 2002**). It takes a number of random trials set by the user and returns the best value. It should be noted that the program does not assign any additional weight to values surrounding the initial guess, like using a Normal distribution would achieve.

2.2 Random Walk

This algorithm starts at random points in the search space, then takes a "step" in a random direction for all variables, then decides to stay at the new point if it is a better answer than the previous solution. This algorithm will arrive at some kind of optima, although it is very prone to coming to rest in local optima. Several restarts of the algorithm are therefore written into the program to increase the odds of finding the global optimum.

2.3 Simulated Annealing

This algorithm was modeled after the metallurgical process of annealing, in which a metal is slowly cooled from a very high temperature in order to have the atoms of the sample settle in the lowest possible energy state (**Brittanica 2010**). Quenching the metal and lowering its temperature rapidly sometimes forces the atoms to settle before they can reach their lowest energy state.

In a similar way, the Simulated Annealing algorithm oscillates around the sample space with an intensity based on the current "temperature" of the system, which is controlled by a cooling

schedule. It also accepts a higher energy state – or a worse solution – with a certain probability based on how far it differs from the current best solution. As the temperature of the system decreases, the solution will fall into a local optimum. Like in the physical process of annealing, the high starting temperature and acceptance of worse solutions allows the algorithm to escape local minima, which is something none of the other methods examined allow for (**Wolfram 2010**).

2.4 Orthogonal Steepest Descent

This algorithm is a stepping searching method. As opposed to other deterministic schemes, OSD is zeroth order, which means that it does not require a derivative to determine the best direction to step in. The method of Orthogonal Steepest Descent samples values from each neighboring data point in a certain step size, steps to the best answer, then narrows the step size until the program reaches an optima (**Raymer 2002**). The method it uses employs no element of chance or the "rand" command, so the program, given the same starting conditions, will always arrive at the same answer in the same exact fashion.

2.5 Algorithm Analysis and Comparison

Two objective functions are used for testing.

One objective function used to test these algorithms was an eight-dimensional downwards-facing parabola with maximum at x = 5, for all eight variables. All tests are run using the programs' default options, and results for stochastic methods were calculated for five trials and averaged. This is a sample function used just to test how accurate the algorithms are.

The test set of initial values (1, 2, 3, 4, 5, 6, 7, 8) will yield several values which should end up being five, and the rest will settle at one end of their ± 20 % range.

The maximum for this data set occurs at [1.2, 2.4, 3.6, 4.8, 5, 5, 5.6, 6.4], and the value of the function at these points is 2.2001e13.

1.) Testing with the 8 variable sample function

<u>Stochastic methods</u>:

Monte Carlo (MCrandomsizing.m), average of five trials:

Average function value:
2.152 e+13 (2.2% error)

Average values of variables:
[1.1864 2.3858 3.4929 4.7581 4.9548 5.2996 5.6664 6.4645] (2.6% error)

Average time elapsed:
16.548 seconds

Random Walk (RWsizing.m), average of five trials:

Average function value:
2.2001 e+13 (~0% error)

Average values of variables:
[1.200 2.400 3.600 4.800 5.000 4.999 5.600 6.400] (0.004% error)

Average time elapsed:
1.861 seconds

Simulated Annealing (annealsizing.m), average of five trials:

Average function value:
2.2001 e+13 (~0% error)

Average values of variables:
[1.200 2.400 3.600 4.800 5.000 4.999 5.600 6.400] (0.004% error)

Average time elapsed:
0.1765 seconds

Deterministic method:

Orthogonal Steepest Descent (OSDsizing.m), only time averaged over five trials:

Function value:
2.1996 e+13 (0.02% error)

Values of variables:
[1.2000 2.4000 3.6000 4.8000 4.9990 4.9994 5.6000 6.4000] (0.0091% error)

Average time elapsed:
0.1168 seconds

2.) Testing with the 4 variable drag function

The other function used for testing is the 'drag in the responsibility of the cabin' function. It is a far more complex function, and requires an order of magnitude more floating-point operations to evaluate. It has four variables, and the program can be run with this new function by replacing the objective function and all its associated function calls and then either inputting '0' for all unused variables or further editing the code to reflect the new number of variables.

Definite values for lower and upper bounds were coded into the function, and these are:
df: [2.7 8]
lf: [20 75]
Aw: [5 15]
Sw: [40 200]

Since "best" values for this function vary, all results will be compared to the algorithm which provides the lowest average function value.

Stochastic methods:

Monte Carlo (MCrandomsizingdrag.m), average of five trials:

Average function value:
2.9659 e+03 (0.6% error)

Average values of variables:
[2.7098 20.0501 5.8667 40.2464 0 0 0 0] (1.19% error)

Average time elapsed:
661.4523 seconds

Random Walk (RWsizingdrag.m), average of five trials:

Average function value:
2.9483 e+03 (~0% error)

Average values of variables:
[2.7000 20.0000 5.3904 40.0000 0 0 0 0] (~0% error)

Average time elapsed:
10.3018 seconds

Simulated Annealing (annealsizingdrag.m), average of five trials:

Average function value:
2.9483 e+03 (~0% error)

Average values of variables:
[2.7000 20.0000 5.3904 40.0000 0 0 0 0] (~0% error)

Average time elapsed:
0.3623 seconds

Deterministic method:

Orthogonal Steepest Descent (OSDsizingdrag.m), only time averaged over five trials:

Function value:
2.9565 e+03 (~0% error)

Values of variables:
[2.7000 20.0000 5.3904 40.0000 0 0 0 0] (~0% error)

Average time elapsed:
6.4173 seconds

3 Summary and Conclusions

Tables 3.1 and 3.2 summarize the results with respect to errors and time elapsed for both test functions.

Table 3.1 Summary of algorithm evaluation results with sample function

		Value error:	Variable error:	Average time:
Algorithm:	Monte Carlo	2.2%	2.6%	16.548 s
	Random Walk	~0%	0.004%	1.861 s
	Simulated Annealing	~0%	0.004%	0.1765 s
	OSD	0.02%	0.0091%	0.1168 s

Table 3.2 Summary of algorithm evaluation results with drag function

		Value error:	Variable error:	Average time:
Algorithm:	Monte Carlo	0.6%	1.19%	661.4523 s
	Random Walk	~0%	~0%	10.3018 s
	Simulated Annealing	~0%	~0%	0.3623 s
	OSD	~0%	~0%	6.4173 s

As is evident from Table 3.1, Orthogonal Steepest Descent seems to be the fastest method which is accurate within the realm of reasonable precision for the sample function. It is about 33% faster than the next fastest method, Simulated Annealing. Random Walk, Simulated Annealing, and Orthogonal Steepest Descent, however, are all very fast and accurate methods. The Random Monte Carlo method is less precise by nature, and experiences a greater error and time elapsed because it requires many more iterations to arrive at reasonable error.

As is evident from Table 3.2, Simulated Annealing is the superior method for the drag function. It arrives at the correct answer while examining the sample space widely, and takes a fraction of the time of any other method.

Each of these algorithms differs slightly in principle. Monte Carlo, although it is slow and inaccurate, samples the entire search space indiscriminately, which suggests it could be very useful to analyze a search space with many local optima. This method could be used as a starting point from which answers are received then plugged into any one of the other methods to refine the answer.

When considering an objective function with many local optima, Simulated Annealing seems to be the best all-purpose algorithm. The way it samples the entire search space randomly while simultaneously falling into an optimum promises to find many of the local optima, and hopefully a global optimum.

When considering an objective function that has either one optimum or is monotonically increasing or decreasing, or one that is not complicated, Orthogonal Steepest Descent is the fastest reliable method. Since time elapsed is exponentially related to variables involved, OSD stops being an efficient method once many more variables are involved.

Because preliminary sizing is a process of aircraft design that does not need to be excessively precise, any of these methods will provide a reasonable starting point for aircraft design. Depending on the complexity of the objective function, several methods can be recommended. The method generally recommended by this project is Simulated Annealing. It is flexible, samples most of the search space, and is very fast and accurate. It should be remarked in this

section, however, that if the objective function in question is to be minimized, the code must be altered so that the line concerning accepting the higher energy state reads:

```
if rand < exp((val-newbestval)/(-1*T))
```

as opposed to:

```
if rand < exp((newbestval-val)/(-1*T))
```

so that the program may work correctly.

In general, the acceptance criteria for new best answers must be altered depending on whether a maximum or minimum is desired.

If the objective function is sufficiently simple, Orthogonal Steepest Descent is the best method. It is as accurate as is necessary for preliminary sizing, and it is the fastest of the methods tested for an eight-variable objective function.

The Random Walk is also a fairly robust method, and with enough restarts, this method always finds an optimum reasonably quickly.

4 List of References

Brittanica 2010 ENCYCLOPEDIA - BRITANNICA ONLINE ENCYCLOPEDIA: *Annealing (heat Treatment)*. – URL: http://www.britannica.com/EBchecked/topic/26292/annealing (2010-07-29)

Kroo 2006 KROO, Ilan: *Techniques for Aircraft Configuration Optimization*. Aircraft Aerodynamics and Design Group, Sept. 2006. – URL: http://adg.stanford.edu/aa241/design/optimization1.html (2010-07-29)

Raymer 2002 RAYMER, D., P.: Enhancing Aircraft Conceptual Design using Multidisciplinary Optimization. Stockholm, Kungliga Tekniska Högskolan Royal Institute of Technology, Department of Aeronautics, Doctoral Thesis, 2002. – ISBN 91-7283-259-2

Wolfram 2010 WOLFRAM MATHWORLD: *Simulated Annealing*. – URL: http://mathworld.wolfram.com/SimulatedAnnealing.html (2010-07-29)

Appendix A

MATLAB Codes

Monte Carlo with Test Function 1 (MCrandomsizing.m)

```
% MCrandomsizing.m
%
% Joseph Yang
% Rice University
% j.yang@rice.edu
% for CARISMA at HAW Hamburg
% 21 June 2010
%
%
% This code applies the Monte Carlo method to optimization of
% preliminary sizing of aircraft design parameters.
%
% usage: [newbestval, bestparams, time]...
%                   = MCrandomsizing(TW, WS, AR, S, TA, TH, F, C, pow)
% where:
%
%           output parameters:
%
%           newbestval = maximum value of optimization function
%           bestparams = best parameters given by optimization function
%
%           (optional output parameters)
%
%           time       = time elapsed to run code
%
%
%           input parameters:
%
%           TW  = thrust to weight ratio
%           WS  = wing sweep
%           AR  = aspect ratio
%           S   = sweep
%           TA  = taper
%           TH  = thickness
%           F   = fineness ratio / slenderness
%           C   = design lift coefficient
%
%           (optional input parameter)
%           pow = power of maximum number of iterations (10^pow)
%                     default is 8



function [newbestval, bestparams, time]...
                   = MCrandomsizing(TW, WS, AR, S, TA, TH, F, Cl, pow)

if nargin == 8;
    pow = 7;                            % set default power
end

if pow > 10
```

```
    pow = 10;                               % set max power limit
end


format long                                 % make rand specific enough

p = 10000;                                  % mesh search space

TWvec = linspace(TW*0.8,TW*1.2,p)';         % define search space here --
WSvec = linspace(WS*0.8,WS*1.2,p)';         % can be defined to include
ARvec = linspace(AR*0.8,AR*1.2,p)';         % definite lower or upper bounds
Svec  = linspace(S*0.8,S*1.2,p)';
TAvec = linspace(TA*0.8,TA*1.2,p)';
THvec = linspace(TH*0.8,TH*1.2,p)';
Fvec  = linspace(F*0.8,F*1.2,p)';
Clvec = linspace(Cl*0.8,Cl*1.2,p)';

A = [TWvec WSvec ARvec Svec TAvec THvec Fvec Clvec];

                                            % make matrix of values



%
%         [ lTW lWS lAR lS lTA lTH lF lCl ]
%         [  .   .   .   .  .   .   .  .  ]
%  A =    [ bTW bWS bAR bS bTA bTH bF bCl ]
%         [  .   .   .   .  .   .   .  .  ]
%         [ uTW uWS uAR uS uTA uTH uF uCl ] p x (# of variables)
%



newbestval = obj(TW,WS,AR,S,TA,TH,F,Cl);    % current best based on
guesses
bestparams = [TW WS AR S TA TH F Cl];

maxiter = 10^pow;                           % set max iterations

tic                                         % start timer

for i = 1:maxiter

TWi = A(ceil(rand*(p)),1);                  % assign random values in
WSi = A(ceil(rand*(p)),2);                  % search space to all
ARi = A(ceil(rand*(p)),3);                  % variables
Si  = A(ceil(rand*(p)),4);
TAi = A(ceil(rand*(p)),5);
THi = A(ceil(rand*(p)),6);
Fi  = A(ceil(rand*(p)),7);
Cli = A(ceil(rand*(p)),8);

val = obj(TWi, WSi, ARi, Si, TAi, THi, Fi, Cli); % evaluate

if val > newbestval                             % is this better?

    newbestval = val;                           % if so, tell program its
                                                % new best
    bestparams = [TWi WSi ARi Si TAi THi Fi Cli];
end
```

```
end

time = toc;                                          % record time

format short

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

function val = obj(TW,WS,AR,S,TA,TH,F,Cl)              % objective function

val = (-(TW-5)^2+50)*(-(WS-5)^2+50)*(-(AR-5)^2+50)*(-(S-5)^2+50)*...
    (-(TA-5)^2+50)*(-(TH-5)^2+50)*(-(F-5)^2+50)*(-(Cl-5)^2+50);
```

Random Walk with Test Function 1 (RWsizing.m)

```
% RWsizing.m
%
% Joseph Yang
% Rice University
% j.yang@rice.edu
% for CARISMA at HAW Hamburg
% 16 June 2010
%
%
% This code applies the Random Walk method to optimization of preliminary
% sizing of aircraft design parameters with the Gaussian random walk
% implemented.
%
%
% usage:    [newbestval, bestparams, time, start]...
%               = RWsizing(TW, WS, AR, S, TA, TH, F, C, run, pow)
%
% where:
%
%           output parameters:
%
%           newbestval = maximum value of optimization function
%                        with each run's value in each row
%           bestparams = best parameters given by optimization function
%                        organized by row into sets of parameters by trial
%
%           (optional output parameters)
%
%           time       = time elapsed to run code
%           start      = location of walk start
%
%
%           input parameters:
%
%           TW  = thrust to weight ratio
%           WS  = ratio of weight to wing area
%           AR  = aspect ratio
%           S   = sweep
%           TA  = taper
%           TH  = thickness
%           F   = fineness ratio / slenderness
%           Cl  = design lift coefficient
%
%           (optional input parameters)
%
%           run = number of runs of algorithm
%                      default is 5
%           pow = power of maximum number of iterations (10^pow)
%                      default is 5, maximum is 8
%
%
%
% example: [bestvals, bestparams, time] = RWsizing(1,2,3,4,5,6,7,8)
%
%          or
%
%          [bestvals, bestparams, time, start]...
%               = RWsizing(1,2,3,4,5,6,7,8,10,5)
```

```
function [bestvals, bestparams, time, start]...
                = RWsizing(TW, WS, AR, S, TA, TH, F, Cl, run, pow)


if nargin == 8;                         % simple setup, call with just
                                        % parameters as variables:
    pow = 5;                            % set default power as 5
                                        % (10^5 iters)
    run = 5;                            % run 5 times for default

end

if nargin == 9;                         % call just how many runs:

    pow = 5;                            % set default power as 5

end

if pow > 8

    pow = 8;                            % cap iterations at 10^8

end


format long                             % make rand specific enough

p = 1000;                               % mesh search space


TWvec = linspace(TW*0.8,TW*1.2,p)';     % define search space here --
WSvec = linspace(WS*0.8,WS*1.2,p)';     % can be defined to include
ARvec = linspace(AR*0.8,AR*1.2,p)';     % definite lower or upper bounds
Svec  = linspace(S *0.8,S *1.2,p)';     % default is to set bounds from
TAvec = linspace(TA*0.8,TA*1.2,p)';     % 80% of original values to 120%
THvec = linspace(TH*0.8,TH*1.2,p)';
Fvec  = linspace(F *0.8,F *1.2,p)';
Clvec = linspace(Cl *0.8,Cl *1.2,p)';

A = [TWvec WSvec ARvec Svec TAvec THvec Fvec Clvec];

                                        % make matrix of values



%
%         [ lTW lWS lAR lS lTA lTH lF lCl ]
%         [  .   .   .   .  .   .   .  .  ]
%  A =    [ bTW bWS bAR bS bTA bTH bF bCl ]
%         [  .   .   .   .  .   .   .  .  ]
%         [ uTW uWS uAR uS uTA uTH uF uCl ] p x 8 (# of variables)
%
```

```
start      = zeros(run,8);              % initialize answer matrices
bestparams = zeros(run,8);
bestvals   = zeros(run,1);

maxiter = 10^pow;                       % set max iterations



tic                                     % start timer


for j = 1:run                           % repeat walk run number of times

TWc = ceil(rand*p);                     % initialize each variable
WSc = ceil(rand*p);                     % randomly in search space
ARc = ceil(rand*p);
Sc  = ceil(rand*p);
TAc = ceil(rand*p);
THc = ceil(rand*p);
Fc  = ceil(rand*p);
Clc = ceil(rand*p);

start(j,:) = [TWc WSc ARc Sc TAc THc Fc Clc];
                                        % write start locations


str1 = 'Trial '; str2 = int2str(j); str3 = ' of '; str4 = int2str(run);
str5 = '. Please wait...';
                                        % write progress bar dialog

str = [str1 str2 str3 str4 str5];       % compile progress bar dialog

waitbar(0,str);                         % display progress bar


for i = 1:maxiter


        r = rand;                       % determine walk direction randomly

        if r < 1/3;                     % if r small, decrease value

            if TWc == 1                 % if at end of search space, stop
                TWcn = TWc;
            else                        % otherwise, step down
                TWcn = TWc-1;
            end

        elseif r < 2/3                  % if r medium, keep same
            TWcn = TWc;

        else                            % if r large, increase value

            if TWc == p                 % if at end of search space, stop
                TWcn = TWc;
            else                        % otherwise, step up
                TWcn = TWc+1;
            end
                                        % (step down, stay, or step up
        end                             %  with equal probability)
```

```
TWin = A(TWcn,1);                % define from value matrix



r = rand;                        % repeat for rest of values

if r < 1/3;

    if WSc == 1
        WScn = WSc;
    else
        WScn = WSc-1;
    end

elseif r < 2/3
    WScn = WSc;

else

    if WSc == p
        WScn = WSc;
    else
        WScn = WSc+1;
    end

end
WSin = A(WScn,2);



r = rand;

if r < 1/3;

    if ARc == 1
        ARcn = ARc;
    else
        ARcn = ARc-1;
    end

elseif r < 2/3
    ARcn = ARc;
else

    if ARc == p
        ARcn = ARc;
    else
        ARcn = ARc+1;
    end

end
ARin = A(ARcn,3);



r = rand;
if r < 1/3;
```

```
        if Sc == 1
            Scn = Sc;
        else
            Scn = Sc-1;
        end

    elseif r < 2/3
        Scn = Sc;

    else

        if Sc == p
            Scn = Sc;
        else
            Scn = Sc+1;
        end

    end
    Sin = A(Scn,4);



    r = rand;
    if r < 1/3;

        if TAc == 1
            TAcn = TAc;
        else
            TAcn = TAc-1;
        end

    elseif r < 2/3
        TAcn = TAc;

    else

        if TAc == p
            TAcn = TAc;
        else
            TAcn = TAc+1;
        end

    end
    TAin = A(TAcn,5);



    r = rand;
    if r < 1/3;

        if THc == 1
            THcn = THc;
        else
            THcn = THc-1;
        end

    elseif r < 2/3
```

```
        THcn = THc;

    else

        if THc == p
            THcn = THc;
        else
            THcn = THc+1;
        end

    end
    THin = A(THcn,6);



    r = rand;
    if r < 1/3;

        if Fc == 1
            Fcn = Fc;
        else
            Fcn = Fc-1;
        end

    elseif r < 2/3
        Fcn = Fc;

    else

        if Fc == p
            Fcn = Fc;
        else
            Fcn = Fc+1;
        end

    end
    Fin = A(Fcn,7);



    r = rand;
    if r < 1/3;

        if Clc == 1
            Clcn = Clc;
        else
            Clcn = Clc-1;
        end

    elseif r < 2/3
        Clcn = Clc;

    else

        if Clc == p
            Clcn = Clc;
        else
            Clcn = Clc+1;
```

```
            end

        end
        Clin = A(Clcn,8);



        val = obj(TWin, WSin, ARin, Sin, TAin, THin, Fin, Clin);
                                        % evaluate


        if i == 1                       % is this the first iteration?
            newbestval = val;           % then use this as a baseline


        elseif val > newbestval         % is this better?


            newbestval = val;           % if so, tell program its new best

            TWi = TWin;                 % update best values for variables
            WSi = WSin;
            ARi = ARin;
            Si  = Sin;
            TAi = TAin;
            THi = THin;
            Fi  = Fin;
            Cli = Clin;

            TWc = TWcn;                 % update matrix location for
            WSc = WScn;                 % variables
            ARc = ARcn;
            Sc  = Scn;
            TAc = TAcn;
            THc = THcn;
            Fc  = Fcn;
            Clc = Clcn;

                                        % (if not, make note and do
                                        %  nothing)
        end



    if (rem(i,maxiter/10) == 0)         % update progress bar every once
        waitbar((i/maxiter))            % in a while
    end


end


bestparams(j,:) = [TWi WSi ARi Si TAi THi Fi Cli];
                                        % write best values at the end
bestvals(j) = newbestval;

end

format short                            % display in command window nicely
```

```
                                                %

time = toc;                                             % record time


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

function val = obj(TW,WS,AR,S,TA,TH,F,Cl)               % objective function

val = (-(TW-5)^2+50)*(-(WS-5)^2+50)*(-(AR-5)^2+50)*(-(S-5)^2+50)*...
    (-(TA-5)^2+50)*(-(TH-5)^2+50)*(-(F-5)^2+50)*(-(Cl-5)^2+50);
```

Simulated Annealing with Test Function 1 (annealsizing.m)

```
% annealsizing.m
%
% Joseph Yang
% Rice University
% j.yang@rice.edu
% for CARISMA at HAW Hamburg
% 14 July 2010
%
%
% This code applies the method of Simulated Annealing to the optimization
% of preliminary sizing of aircraft design parameters.
%
%
% usage:    [newbestval, bestparams, time, iter]...
%               = annealsizing(TW, WS, AR, S, TA, TH, F, C, run, dT, stop)
%
% where:
%
%           output parameters:
%
%           newbestval = maximum value of optimization function
%                        with each run's value in each row
%           bestparams = best parameters given by optimization function
%                        organized by row into sets of parameters by trial
%
%           (optional output parameters)
%
%           time       = time elapsed to run code
%           iter       = number of iterations taken
%
%
%           input parameters:
%
%           TW   = thrust to weight ratio
%           WS   = ratio of weight to wing area
%           AR   = aspect ratio
%           S    = sweep
%           TA   = taper
%           TH   = thickness
%           F    = fineness ratio / slenderness
%           Cl   = design lift coefficient
%
%           (optional input parameters)
%
%           dT   = ratio of temperature of next iteration to temperature of
%                  previous iteration (cooling schedule)
%                      default is 0.98
%           run  = number of runs of algorithm
%                      default is 5
%           stop = defines stopping temperature as 10^-stop
%                      default is 10
%
%
% example: [bestvals, bestparams] = annealsizing(1,2,3,4,5,6,7,8)
%
%          or
%
%          [bestvals, bestparams, time, iter]...
```

```
%               = annealsizing(1,2,3,4,5,6,7,8,0.95,10,10)



function [bestvals, bestparams, time, iter]...
                = annealsizing(TW, WS, AR, S, TA, TH, F, Cl, dT, run, stop)


if nargin == 8;                         % simple setup, call with just
                                        % parameters as variables:

    stop = 10;                          % set default minimum temp

    run  = 5;                           % run 5 times for default

    dT = 0.99;                          % set default cooling schedule

end

if nargin == 9;                         % call just how many runs:

    stop = 10;                          % set default minimum temp

    run = 5;                           % set default cooling schedule

end

if nargin == 10;                        % call just how many runs and dT:

    stop = 10;                          % set default minimum temp

end

if stop > 12

    stop = 12;                          % cap minimum temperature at ~eps

end



format long                             % make rand specific enough

p = 1000;                               % mesh search space


TWvec = linspace(TW*0.8,TW*1.2,p)';     % define search space here --
WSvec = linspace(WS*0.8,WS*1.2,p)';     % can be defined to include
ARvec = linspace(AR*0.8,AR*1.2,p)';     % definite lower or upper bounds
Svec  = linspace(S *0.8,S *1.2,p)';     % default is to set bounds from
TAvec = linspace(TA*0.8,TA*1.2,p)';     % 80% of original values to 120%
THvec = linspace(TH*0.8,TH*1.2,p)';
Fvec  = linspace(F *0.8,F *1.2,p)';
Clvec = linspace(Cl*0.8,Cl*1.2,p)';

A = [TWvec WSvec ARvec Svec TAvec THvec Fvec Clvec];

                                        % make matrix of values
```

```
%
%         [ lTW lWS lAR lS lTA lTH lF lCl ]
%         [  .   .   .   .  .   .   .  .  ]
%  A =    [ bTW bWS bAR bS bTA bTH bF bCl ]
%         [  .   .   .   .  .   .   .  .  ]
%         [ uTW uWS uAR uS uTA uTH uF uCl ] p x 8 (# of variables)
%


bestparams = zeros(run,8);
bestvals   = zeros(run,1);


tic                                     % start timer


for j = 1:run                           % repeat walk run number of times

TWc = 0.5*p;                            % initialize each variable at
WSc = 0.5*p;                            % initial guess
ARc = 0.5*p;
Sc  = 0.5*p;
TAc = 0.5*p;
THc = 0.5*p;
Fc  = 0.5*p;
Clc = 0.5*p;



str1 = 'Trial '; str2 = int2str(j); str3 = ' of '; str4 = int2str(run);
str5 = '. Please wait...';
                                        % write progress bar dialog

str = [str1 str2 str3 str4 str5];       % compile progress bar dialog

waitbar(0,str);                         % display progress bar

maxiter = 2292;                         % how many iterations the program
                                        % at dT = 0.99

T = 1;

iter = 0;


    while T > 10^(-1*stop)

            iter = iter + 1;

            tempmove = ceil(T*500);         % 500 is empirically derived

            move = ceil(tempmove*rand);     % determine walk direction and
            r = rand;                       % magnitude randomly based on
                                            % current temperature

            if r < 1/3;                     % if r small, decrease value
                TWcn = TWc-move;
```

```
elseif r < 2/3                  % if r medium, keep same
    TWcn = TWc;
else                            % if r large, increase value
    TWcn = TWc+move;            % (step down, stay, or step up
end                             %  with equal probability)


if TWcn > p                     % stop from going outside of
    TWcn = p;                   % boundaries
elseif TWcn < 1
    TWcn = 1;
end

TWin = A(TWcn,1);               % define from value matrix


move = ceil(tempmove*rand);     %
r = rand;                       % repeat for other variables

if r < 1/3;
    WScn = WSc-move;
elseif r < 2/3
    WScn = WSc;
else
    WScn = WSc+move;
end


if WScn > p
    WScn = p;
elseif WScn < 1
    WScn = 1;
end

WSin = A(WScn,2);


move = ceil(tempmove*rand);
r = rand;

if r < 1/3;
    ARcn = ARc-move;
elseif r < 2/3
    ARcn = ARc;
else
    ARcn = ARc+move;
end


if ARcn > p
    ARcn = p;
elseif ARcn < 1
    ARcn = 1;
end

ARin = A(ARcn,3);


move = ceil(tempmove*rand);
r = rand;
```

```
if r < 1/3;
    Scn = Sc-move;
elseif r < 2/3
    Scn = Sc;
else
    Scn = Sc+move;
end


if Scn > p
    Scn = p;
elseif Scn < 1
    Scn = 1;
end

Sin = A(Scn,4);


move = ceil(tempmove*rand);
r = rand;

if r < 1/3;
    TAcn = TAc-move;
elseif r < 2/3
    TAcn = TAc;
else
    TAcn = TAc+move;
end


if TAcn > p
    TAcn = p;
elseif TAcn < 1
    TAcn = 1;
end

TAin = A(TAcn,5);


move = ceil(tempmove*rand);
r = rand;

if r < 1/3;
    THcn = THc-move;
elseif r < 2/3
    THcn = THc;
else
    THcn = THc+move;
end


if THcn > p
    THcn = p;
elseif THcn < 1
    THcn = 1;
end

THin = A(THcn,6);
```

```
            move = ceil(tempmove*rand);
            r = rand;

            if r < 1/3;
                Fcn = Fc-move;
            elseif r < 2/3
                Fcn = Fc;
            else
                Fcn = Fc+move;
            end


            if Fcn > p
                Fcn = p;
            elseif Fcn < 1
                Fcn = 1;
            end

            Fin = A(Fcn,7);


            move = ceil(tempmove*rand);
            r = rand;

            if r < 1/3;
                Clcn = Clc-move;
            elseif r < 2/3
                Clcn = Clc;
            else
                Clcn = Clc+move;
            end


            if Clcn > p
                Clcn = p;
            elseif Clcn < 1
                Clcn = 1;
            end

            Clin = A(Clcn,8);



            val = obj(TWin, WSin, ARin, Sin, TAin, THin, Fin, Clin);
                                            % evaluate


            if iter == 1                    % is this the first iteration?
                newbestval = val;           % then use this as a baseline


            elseif val > newbestval         % is this better?


                newbestval = val;           % if so, tell program its new
%best

                TWi = TWin;                 % update best values for
%variables
```

```
                WSi = WSin;
                ARi = ARin;
                Si  = Sin;
                TAi = TAin;
                THi = THin;
                Fi  = Fin;
                Cli = Clin;

                TWc = TWcn;                 % update matrix location for
                WSc = WScn;                 % variables
                ARc = ARcn;
                Sc  = Scn;
                TAc = TAcn;
                THc = THcn;
                Fc  = Fcn;
                Clc = Clcn;

            else

                if rand < exp((newbestval-val)/(-1*T))

                                            % accept worse solution with
                                            % probability exp(-deltaVal/T)
                                            % so that the program may get
                                            % out of local minima

                newbestval = val;           % if so, tell program new best

                TWi = TWin;                 % update best values
                WSi = WSin;
                ARi = ARin;
                Si  = Sin;
                TAi = TAin;
                THi = THin;
                Fi  = Fin;
                Cli = Clin;

                TWc = TWcn;                 % update matrix location for
                WSc = WScn;                 % variables
                ARc = ARcn;
                Sc  = Scn;
                TAc = TAcn;
                THc = THcn;
                Fc  = Fcn;
                Clc = Clcn;

                end

            end

        if (rem(iter,maxiter/10) == 0)      % update progress bar every
            waitbar((iter/maxiter))         % once in a while
        end

        T = dT*T;

    end


bestparams(j,:) = [TWi WSi ARi Si TAi THi Fi Cli];
```

```
                                                % write best values at the end
bestvals(j) = newbestval;

end

format short                                    % display nicely


time = toc;                                     % record time


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

function val = obj(TW,WS,AR,S,TA,TH,F,Cl)    % objective function

val = (-(TW-5)^2+50)*(-(WS-5)^2+50)*(-(AR-5)^2+50)*(-(S-5)^2+50)*...
    (-(TA-5)^2+50)*(-(TH-5)^2+50)*(-(F-5)^2+50)*(-(Cl-5)^2+50);
```

Orthogonal Steepest Descent with Test Function 1 (OSDsizing.m)

```
% OSDsizing.m
%
% Joseph Yang
% Rice University
% j.yang@rice.edu
% for CARISMA at HAW Hamburg
% 20 July 2010
%
%
% This code applies the Orthogonal Steepest Descent method to optimization
% of preliminary sizing of aircraft design parameters.
%
%
% usage:    [newbestval, bestparams, time]...
%               = OSDsizing(TW, WS, AR, S, TA, TH, F, Cl)
%
% where:
%
%           output parameters:
%
%           newbestval = maximum value of optimization function
%                        with each run's value in each row
%           bestparams = best parameters given by optimization function
%                        organized by row into sets of parameters by trial
%
%           (optional output parameters)
%
%           time       = time elapsed to run code
%           iter       = iterations required to obtain answer
%
%
%           input parameters:
%
%           TW  = thrust to weight ratio
%           WS  = ratio of weight to wing area
%           AR  = aspect ratio
%           S   = sweep
%           TA  = taper
%           TH  = thickness
%           F   = fineness ratio / slenderness
%           Cl  = design lift coefficient
%
%
%
% example: [bestvals, bestparams] = OSDsizing(1,2,3,4,5,6,7,8)
%
%          or
%
%          [bestvals, bestparams, time]...
%               = OSDsizing(1,2,3,4,5,6,7,8,10,5)

function [bestval, bestparams, time, iter]...
                   = OSDsizing(TW, WS, AR, S, TA, TH, F, Cl)


tic

iter = 0;                                % initialize iteration counter
```

```
p = 1000;                                   % mesh search space

step = p*0.2;


TWvec = linspace(TW*0.8,TW*1.2,p)';         % define search space here --
WSvec = linspace(WS*0.8,WS*1.2,p)';         % can be defined to include
ARvec = linspace(AR*0.8,AR*1.2,p)';         % definite lower or upper bounds
Svec  = linspace(S *0.8,S *1.2,p)';         % default is to set bounds from
TAvec = linspace(TA*0.8,TA*1.2,p)';         % 80% of original values to 120%
THvec = linspace(TH*0.8,TH*1.2,p)';
Fvec  = linspace(F *0.8,F *1.2,p)';
Clvec = linspace(Cl*0.8,Cl*1.2,p)';

A = [TWvec WSvec ARvec Svec TAvec THvec Fvec Clvec];

TWc = 0.5*p;                        % initialize each variable
WSc = 0.5*p;                        % in the middle of search space
ARc = 0.5*p;
Sc  = 0.5*p;
TAc = 0.5*p;
THc = 0.5*p;
Fc  = 0.5*p;
Clc = 0.5*p;

finished = 0;

newbestval = obj(TW,WS,AR,S,TA,TH,F,Cl);
bestval = obj(TW,WS,AR,S,TA,TH,F,Cl);
bestparams = [TW,WS,AR,S,TA,TH,F,Cl];


while finished == 0;                % run until smallest interval reached

changed = 0;

for a = 1:3                                 % loop over every parameter
    apos = (TWc + (a-2)*step);              % determine step size

    if apos > p
        apos = p;
    elseif apos < 1
        apos = 1;
    end

    aval = A(apos,1);

    for b = 1:3

        bpos = (WSc + (b-2)*step);

        if bpos > p
            bpos = p;
        elseif bpos < 1
            bpos = 1;
        end

        bval = A(bpos,2);
```

```
for c = 1:3

    cpos = (ARc + (c-2)*step);

    if cpos > p
        cpos = p;
    elseif cpos < 1
        cpos = 1;
    end

    cval = A(cpos,3);

    for d = 1:3

        dpos = (Sc + (d-2)*step);

        if dpos > p
            dpos = p;
        elseif dpos < 1
            dpos = 1;
        end

        dval = A(dpos,4);

        for e = 1:3

            epos = (TAc + (e-2)*step);

            if epos > p
                epos = p;
            elseif epos < 1
                epos = 1;
            end

            eval = A(epos,5);

            for f = 1:3

                fpos = (THc + (f-2)*step);

                if fpos > p
                    fpos = p;
                elseif fpos < 1
                    fpos = 1;
                end

                fval = A(fpos,6);

                for g = 1:3

                    gpos = (Fc + (g-2)*step);

                    if gpos > p
                        gpos = p;
                    elseif gpos < 1
                        gpos = 1;
                    end

                    gval = A(gpos,7);
```

```
                            for h = 1:3

                                hpos = (Clc + (h-2)*step);

                                if hpos > p
                                    hpos = p;
                                elseif hpos < 1
                                    hpos = 1;
                                end

                                hval = A(hpos,8);

                                % evaluate:

                                bestval = obj(aval,bval,cval,dval,...
                                    eval,fval,gval,hval);

                                if bestval > newbestval
                                    newbestval = bestval;
                                    bestparams = [aval bval cval dval...
                                        eval fval gval hval];
                                    TWi = apos;
                                    WSi = bpos;     % if better, update
                                    ARi = cpos;
                                    Si  = dpos;
                                    TAi = epos;
                                    THi = fpos;
                                    Fi  = gpos;
                                    Cli = hpos;
                                    changed = 1;
                                end

                                iter = iter + 1;    % count

                            end
                        end
                    end
                end
            end
        end
    end
end

if changed == 1;                                    % if better soln is
                                                    % found, update
TWc = TWi;
WSc = WSi;
ARc = ARi;
Sc  = Si;
TAc = TAi;
THc = THi;
Fc  = Fi;
Clc = Cli;

elseif changed == 0;                                % if not, stop

    if step == 1;                                   % if smallest step
        finished = 1;                               % already taken, stop
        time = toc;
```

```
    end

    step = ceil(step*0.5);                          % otherwise, refine
                                                    % search


end

end

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

function val = obj(TW,WS,AR,S,TA,TH,F,Cl)              % objective function

val = (-(TW-5)^2+50)*(-(WS-5)^2+50)*(-(AR-5)^2+50)*(-(S-5)^2+50)*...
    (-(TA-5)^2+50)*(-(TH-5)^2+50)*(-(F-5)^2+50)*(-(Cl-5)^2+50);
```

Monte Carlo with Test Function 2 (MCrandomsizingdrag.m)

```
% MCrandomsizingdrag.m
%
% Joseph Yang
% Rice University
% j.yang@rice.edu
% for CARISMA at HAW Hamburg
% 21 June 2010
%
%
% This code applies the Monte Carlo method to optimization of
% preliminary sizing of aircraft design parameters.
%
% usage: [newbestval, bestparams, time]...
%                   = MCrandomsizingdrag(TW, WS, AR, S, TA, TH, F, C, pow)
% where:
%
%           output parameters:
%
%           newbestval = maximum value of optimization function
%           bestparams = best parameters given by optimization function
%
%           (optional output parameters)
%
%           time       = time elapsed to run code
%
%
%           input parameters:
%
%           TW  = thrust to weight ratio
%           WS  = wing sweep
%           AR  = aspect ratio
%           S   = sweep
%           TA  = taper
%           TH  = thickness
%           F   = fineness ratio / slenderness
%           C   = design lift coefficient
%
%           (optional input parameter)
%           pow = power of maximum number of iterations (10^pow)
%                     default is 8



function [newbestval, bestparams, time]...
                    = MCrandomsizingdrag(TW, WS, AR, S, TA, TH, F, Cl, pow)

if nargin == 8;
    pow = 7;                            % set default power
end

if pow > 10
    pow = 10;                           % set max power limit
end


format long                             % make rand specific enough

p = 10000;                              % mesh search space
```

```
TWvec = linspace(2.7, 8 , p)';     % define search space here --
WSvec = linspace(20 , 25, p)';     % can be defined to include
ARvec = linspace(5  , 15, p)';     % definite lower or upper bounds
Svec  = linspace(40 , 200,p)';     % default is to set bounds from
TAvec = linspace(TA*0.8,TA*1.2,p)';
THvec = linspace(TH*0.8,TH*1.2,p)';
Fvec  = linspace(F*0.8,F*1.2,p)';
Clvec = linspace(Cl*0.8,Cl*1.2,p)';

A = [TWvec WSvec ARvec Svec TAvec THvec Fvec Clvec];

                                        % make matrix of values



%
%         [ lTW lWS lAR lS lTA lTH lF lCl ]
%         [  .   .   .   .  .   .   .  .  ]
%  A =    [ bTW bWS bAR bS bTA bTH bF bCl ]
%         [  .   .   .   .  .   .   .  .  ]
%         [ uTW uWS uAR uS uTA uTH uF uCl ] p x (# of variables)
%



newbestval = drag(TW,WS,AR,S);     % current best based on guesses
bestparams = [TW WS AR S TA TH F Cl];

maxiter = 10^pow;                               % set max iterations

tic                                             % start timer

for i = 1:maxiter

TWi = A(ceil(rand*(p)),1);                      % assign random values in
WSi = A(ceil(rand*(p)),2);                      % search space to all
ARi = A(ceil(rand*(p)),3);                      % variables
Si  = A(ceil(rand*(p)),4);
TAi = A(ceil(rand*(p)),5);
THi = A(ceil(rand*(p)),6);
Fi  = A(ceil(rand*(p)),7);
Cli = A(ceil(rand*(p)),8);

val = drag(TWi, WSi, ARi, Si); % evaluate

if val < newbestval                                % is this better?

    newbestval = val;                              % if so, tell program its
                                                   % new best
    bestparams = [TWi WSi ARi Si TAi THi Fi Cli];
end

end

time = toc;                                        % record time

format short
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [DragF] = drag(df,lf,Aw,Sw);

M=0.4;
H=4500;
nPAX=70;

V=130;
g=9.807;
L=0.0065;
Tzero=288.15;
T=Tzero-L*H;
RHOzero=1.225;
RHO=RHOzero*(1-2.2558/10^5*H)^4.25588;
a=20.0468*sqrt(T);
MIU=1.458/10^6*T^1.5/(T+110.4);
NIU=MIU/RHO;
q=0.5*RHO*V^2;
Rek=V/NIU;

%Input data: ATR 72
Cmac=2.2345; %it can be variabile
CV=0.103;
CH=1.17;
ThicknessRatioH=0.09;
ThicknessRatioV=0.12;
LAMBDAh=0.6;
LAMBDAv=0.6;
TAUh=0.7;
TAUv=0.7;
PHImH=7.531;
PHImV=24.607;
PHIh=7;
e=0.75;
PHIv=24;
xt=0.3;
kcabin=1.1;

%Input equations
b=sqrt(Aw*Sw);
Av=1.6;
Ah=0.38*Aw;
Sv=CV*Sw*b/lf/0.5;
Sh=CH*Sw*Cmac/lf/0.5;
bv=sqrt(Av*Sv);
bh=sqrt(Ah*Sh);
% lbug=df*1.7;
Ref=Rek*lf;
% lheck=df*3.5;
%lcabin=nPAX/nSA*kcabin
%Sf=df*lcabin+df*lbug/2+df*lheck/2

%Drag in the responsability of the cabin
k=1/pi()/Aw/e;

%Zero lift drag fuselage
FFfus=1+df^3*60/lf^3+lf/df/400;
SwetF=pi()*df*lf*((1-df*2/lf)^(2/3))*(1+df^2/lf^2);
Cffus=0.455/log10(Ref)^2.58/(1+0.144*M^2)^0.65;
```

```
Cd0fus=Cffus*FFfus*SwetF/Sw;
D0fus=q*Sw*Cd0fus;

%Zero lift drag horizontal empenage
Reh=Rek*bh;
Cfhorlaminar=1.328/sqrt(Reh);
Cfhorturbulent=0.455/(log10(Reh))^2.58/(1+0.144*M^2)^0.65;
Cfhor=20/100*Cfhorlaminar+80/100*Cfhorturbulent;
FFhor=(1+0.6/xt*ThicknessRatioH+100*ThicknessRatioH^4)*1.34*M^0.18*cos(PHIm
H*pi()/180)^0.28;
Qhor=1.04;
SwetH=2*Sh*(1+0.25*ThicknessRatioH*(1+TAUh*LAMBDAh)/(1+LAMBDAh));
Cd0hor=Cfhor*FFhor*Qhor*SwetH/Sw;
D0hor=q*Sw*Cd0hor;

%Zero lift drag vertical empennage;
Rev=Rek*bv;
Cfverlaminar=1.328/sqrt(Rev);
Cfverturbulent=0.455/(log10(Rev))^2.58/(1+0.144*M^2)^0.65;
Cfver=20/100*Cfverlaminar+80/100*Cfverturbulent;
FFver=(1+0.6/xt*ThicknessRatioV+100*ThicknessRatioV^4)*1.34*M^0.18*cos(PHIm
V*pi()/180)^0.28;
Qver=1.04;
SwetV=2*Sv*(1+0.25*ThicknessRatioV*(1+TAUv*LAMBDAv)/(1+LAMBDAv));
Cd0ver=Cfver*FFver*Qver*SwetV/Sw;
D0ver=q*Sw*Cd0ver;

D0=D0fus+D0ver+D0hor;

%Induced drag;
%Fuselage mass;
Vd=(M+0.09)*a;
mf=0.23*sqrt(Vd*lf/4/df)*SwetF^1.2;

%Empenage mass;
mh=Sh*(62*Sh^0.2*Vd/1000/cos(PHIh)-2.5);
mv=Sv*(62*Sv^0.2*Vd/1000/cos(PHIv)-2.5);

%Lift coeficient;
CIF=(mf+mh+mv)*g/q/Sw;
Cdi=k*CIF^2;
Di=q*Sw*Cdi;

% Objective function
DragF=D0+Di;
```

Random Walk with Test Function 2 (RWsizingdrag.m)

```
% RWsizingdrag.m
%
% Joseph Yang
% Rice University
% j.yang@rice.edu
% for CARISMA at HAW Hamburg
% 16 June 2010
%
%
% This code applies the Random Walk method to optimization of preliminary
% sizing of aircraft design parameters with the Gaussian random walk
% implemented.
%
%
% usage:    [newbestval, bestparams, time, start]...
%               = RWsizingdrag(TW, WS, AR, S, TA, TH, F, C, run, pow)
%
% where:
%
%           output parameters:
%
%           newbestval = maximum value of optimization function
%                        with each run's value in each row
%           bestparams = best parameters given by optimization function
%                        organized by row into sets of parameters by trial
%
%           (optional output parameters)
%
%           time       = time elapsed to run code
%           start      = location of walk start
%
%
%           input parameters:
%
%           TW  = thrust to weight ratio
%           WS  = ratio of weight to wing area
%           AR  = aspect ratio
%           S   = sweep
%           TA  = taper
%           TH  = thickness
%           F   = fineness ratio / slenderness
%           Cl  = design lift coefficient
%
%           (optional input parameters)
%
%           run = number of runs of algorithm
%                      default is 5
%           pow = power of maximum number of iterations (10^pow)
%                      default is 5, maximum is 8
%
%
%
% example: [bestvals, bestparams, time] = RWsizingdrag(1,2,3,4,5,6,7,8)
%
%          or
%
%          [bestvals, bestparams, time, start]...
%               = RWsizingdrag(1,2,3,4,5,6,7,8,10,5)
```

```
function [bestvals, bestparams, time, start]...
                = RWsizingdrag(TW, WS, AR, S, TA, TH, F, Cl, run, pow)


if nargin == 8;                             % simple setup, call with just
                                            % parameters as variables:
    pow = 5;                                % set default power as 5
                                            % (10^5 iters)
    run = 5;                                % run 5 times for default

end

if nargin == 9;                             % call just how many runs:

    pow = 5;                                % set default power as 5

end

if pow > 8

    pow = 8;                                % cap iterations at 10^8

end


format long                                 % make rand specific enough

p = 1000;                                   % mesh search space


TWvec = linspace(2.7, 8 , p)';      % define search space here --
WSvec = linspace(20 , 25, p)';      % can be defined to include
ARvec = linspace(5  , 15, p)';      % definite lower or upper bounds
Svec  = linspace(40 , 200,p)';      % default is to set bounds from
TAvec = linspace(TA*0.8,TA*1.2,p)';     % 80% of original values to 120%
THvec = linspace(TH*0.8,TH*1.2,p)';
Fvec  = linspace(F *0.8,F *1.2,p)';
Clvec = linspace(Cl *0.8,Cl *1.2,p)';

A = [TWvec WSvec ARvec Svec TAvec THvec Fvec Clvec];

                                            % make matrix of values



%
%         [ lTW lWS lAR lS lTA lTH lF lCl ]
%         [  .   .   .   .  .   .   .  .  ]
%  A =    [ bTW bWS bAR bS bTA bTH bF bCl ]
%         [  .   .   .   .  .   .   .  .  ]
%         [ uTW uWS uAR uS uTA uTH uF uCl ] p x 8 (# of variables)
%
```

```
start      = zeros(run,8);              % initialize answer matrices
bestparams = zeros(run,8);
bestvals   = zeros(run,1);

maxiter = 10^pow;                       % set max iterations



tic                                     % start timer


for j = 1:run                           % repeat walk run number of times

TWc = ceil(rand*p);                     % initialize each variable
WSc = ceil(rand*p);                     % randomly in search space
ARc = ceil(rand*p);
Sc  = ceil(rand*p);
TAc = ceil(rand*p);
THc = ceil(rand*p);
Fc  = ceil(rand*p);
Clc = ceil(rand*p);

start(j,:) = [TWc WSc ARc Sc TAc THc Fc Clc];
                                        % write start locations


str1 = 'Trial '; str2 = int2str(j); str3 = ' of '; str4 = int2str(run);
str5 = '. Please wait...';
                                        % write progress bar dialog

str = [str1 str2 str3 str4 str5];       % compile progress bar dialog

waitbar(0,str);                         % display progress bar


for i = 1:maxiter


        r = rand;                       % determine walk direction randomly

        if r < 1/3;                     % if r small, decrease value

            if TWc == 1                 % if at end of search space, stop
                TWcn = TWc;
            else                        % otherwise, step down
                TWcn = TWc-1;
            end

        elseif r < 2/3                  % if r medium, keep same
            TWcn = TWc;

        else                            % if r large, increase value

            if TWc == p                 % if at end of search space, stop
                TWcn = TWc;
            else                        % otherwise, step up
                TWcn = TWc+1;
            end
                                        % (step down, stay, or step up
        end                             %  with equal probability)
```

```
TWin = A(TWcn,1);                % define from value matrix



r = rand;                        % repeat for rest of values

if r < 1/3;

    if WSc == 1
        WScn = WSc;
    else
        WScn = WSc-1;
    end

elseif r < 2/3
    WScn = WSc;

else

    if WSc == p
        WScn = WSc;
    else
        WScn = WSc+1;
    end

end
WSin = A(WScn,2);



r = rand;

if r < 1/3;

    if ARc == 1
        ARcn = ARc;
    else
        ARcn = ARc-1;
    end

elseif r < 2/3
    ARcn = ARc;
else

    if ARc == p
        ARcn = ARc;
    else
        ARcn = ARc+1;
    end

end
ARin = A(ARcn,3);



r = rand;
if r < 1/3;
```

```
        if Sc == 1
            Scn = Sc;
        else
            Scn = Sc-1;
        end

    elseif r < 2/3
        Scn = Sc;

    else

        if Sc == p
            Scn = Sc;
        else
            Scn = Sc+1;
        end

    end
    Sin = A(Scn,4);



    r = rand;
    if r < 1/3;

        if TAc == 1
            TAcn = TAc;
        else
            TAcn = TAc-1;
        end

    elseif r < 2/3
        TAcn = TAc;

    else

        if TAc == p
            TAcn = TAc;
        else
            TAcn = TAc+1;
        end

    end
    TAin = A(TAcn,5);



    r = rand;
    if r < 1/3;

        if THc == 1
            THcn = THc;
        else
            THcn = THc-1;
        end

    elseif r < 2/3
```

```
        THcn = THc;

    else

        if THc == p
            THcn = THc;
        else
            THcn = THc+1;
        end

    end
    THin = A(THcn,6);



    r = rand;
    if r < 1/3;

        if Fc == 1
            Fcn = Fc;
        else
            Fcn = Fc-1;
        end

    elseif r < 2/3
        Fcn = Fc;

    else

        if Fc == p
            Fcn = Fc;
        else
            Fcn = Fc+1;
        end

    end
    Fin = A(Fcn,7);



    r = rand;
    if r < 1/3;

        if Clc == 1
            Clcn = Clc;
        else
            Clcn = Clc-1;
        end

    elseif r < 2/3
        Clcn = Clc;

    else

        if Clc == p
            Clcn = Clc;
        else
            Clcn = Clc+1;
```

```
            end

        end
        Clin = A(Clcn,8);



        val = drag(TWin, WSin, ARin, Sin);
                                        % evaluate


        if i == 1                       % is this the first iteration?
            newbestval = val;           % then use this as a baseline


        elseif val < newbestval         % is this better?


            newbestval = val;           % if so, tell program its new best

            TWi = TWin;                 % update best values for variables
            WSi = WSin;
            ARi = ARin;
            Si  = Sin;
            TAi = TAin;
            THi = THin;
            Fi  = Fin;
            Cli = Clin;

            TWc = TWcn;                 % update matrix location for
            WSc = WScn;                 % variables
            ARc = ARcn;
            Sc  = Scn;
            TAc = TAcn;
            THc = THcn;
            Fc  = Fcn;
            Clc = Clcn;

                                        % (if not, make note and do
                                        %  nothing)
        end



    if (rem(i,maxiter/10) == 0)         % update progress bar every once
        waitbar((i/maxiter))            % in a while
    end


end


bestparams(j,:) = [TWi WSi ARi Si TAi THi Fi Cli];
                                        % write best values at the end
bestvals(j) = newbestval;

end

format short                            % display in command window nicely
```

```
                                                %

time = toc;                                             % record time


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

function [DragF] = drag(df,lf,Aw,Sw);

M=0.4;
H=4500;
nPAX=70;

V=130;
g=9.807;
L=0.0065;
Tzero=288.15;
T=Tzero-L*H;
RHOzero=1.225;
RHO=RHOzero*(1-2.2558/10^5*H)^4.25588;
a=20.0468*sqrt(T);
MIU=1.458/10^6*T^1.5/(T+110.4);
NIU=MIU/RHO;
q=0.5*RHO*V^2;
Rek=V/NIU;

%Input data: ATR 72
Cmac=2.2345; %it can be variabile
CV=0.103;
CH=1.17;
ThicknessRatioH=0.09;
ThicknessRatioV=0.12;
LAMBDAh=0.6;
LAMBDAv=0.6;
TAUh=0.7;
TAUv=0.7;
PHImH=7.531;
PHImV=24.607;
PHIh=7;
e=0.75;
PHIv=24;
xt=0.3;
kcabin=1.1;

%Input equations
b=sqrt(Aw*Sw);
Av=1.6;
Ah=0.38*Aw;
Sv=CV*Sw*b/lf/0.5;
Sh=CH*Sw*Cmac/lf/0.5;
bv=sqrt(Av*Sv);
bh=sqrt(Ah*Sh);
% lbug=df*1.7;
Ref=Rek*lf;
% lheck=df*3.5;
%lcabin=nPAX/nSA*kcabin
%Sf=df*lcabin+df*lbug/2+df*lheck/2

%Drag in the responsability of the cabin
k=1/pi()/Aw/e;
```

```
%Zero lift drag fuselage
FFfus=1+df^3*60/lf^3+lf/df/400;
SwetF=pi()*df*lf*((1-df*2/lf)^(2/3))*(1+df^2/lf^2);
Cffus=0.455/log10(Ref)^2.58/(1+0.144*M^2)^0.65;
Cd0fus=Cffus*FFfus*SwetF/Sw;
D0fus=q*Sw*Cd0fus;

%Zero lift drag horizontal empenage
Reh=Rek*bh;
Cfhorlaminar=1.328/sqrt(Reh);
Cfhorturbulent=0.455/(log10(Reh))^2.58/(1+0.144*M^2)^0.65;
Cfhor=20/100*Cfhorlaminar+80/100*Cfhorturbulent;
FFhor=(1+0.6/xt*ThicknessRatioH+100*ThicknessRatioH^4)*1.34*M^0.18*cos(PHIm
H*pi()/180)^0.28;
Qhor=1.04;
SwetH=2*Sh*(1+0.25*ThicknessRatioH*(1+TAUh*LAMBDAh)/(1+LAMBDAh));
Cd0hor=Cfhor*FFhor*Qhor*SwetH/Sw;
D0hor=q*Sw*Cd0hor;

%Zero lift drag vertical empennage;
Rev=Rek*bv;
Cfverlaminar=1.328/sqrt(Rev);
Cfverturbulent=0.455/(log10(Rev))^2.58/(1+0.144*M^2)^0.65;
Cfver=20/100*Cfverlaminar+80/100*Cfverturbulent;
FFver=(1+0.6/xt*ThicknessRatioV+100*ThicknessRatioV^4)*1.34*M^0.18*cos(PHIm
V*pi()/180)^0.28;
Qver=1.04;
SwetV=2*Sv*(1+0.25*ThicknessRatioV*(1+TAUv*LAMBDAv)/(1+LAMBDAv));
Cd0ver=Cfver*FFver*Qver*SwetV/Sw;
D0ver=q*Sw*Cd0ver;

D0=D0fus+D0ver+D0hor;

%Induced drag;
%Fuselage mass;
Vd=(M+0.09)*a;
mf=0.23*sqrt(Vd*lf/4/df)*SwetF^1.2;

%Empenage mass;
mh=Sh*(62*Sh^0.2*Vd/1000/cos(PHIh)-2.5);
mv=Sv*(62*Sv^0.2*Vd/1000/cos(PHIv)-2.5);

%Lift coeficient;
CIF=(mf+mh+mv)*g/q/Sw;
Cdi=k*CIF^2;
Di=q*Sw*Cdi;

% Objective function
DragF=D0+Di;
```

Simulated Annealing with Test Function 2 (annealsizingdrag.m)

```
% annealsizingdrag.m
%
% Joseph Yang
% Rice University
% j.yang@rice.edu
% for CARISMA at HAW Hamburg
% 14 July 2010
%
%
% This code applies the method of Simulated Annealing to the optimization
% of preliminary sizing of aircraft design parameters.
%
%
% usage:    [newbestval, bestparams, time, iter]...
%            = annealsizingdrag(TW, WS, AR, S, TA, TH, F, C, dT, run, stop)
%
% where:
%
%           output parameters:
%
%           newbestval = maximum value of optimization function
%                        with each run's value in each row
%           bestparams = best parameters given by optimization function
%                        organized by row into sets of parameters by trial
%
%           (optional output parameters)
%
%           time       = time elapsed to run code
%           iter       = number of iterations taken
%
%
%           input parameters:
%
%           TW   = thrust to weight ratio
%           WS   = ratio of weight to wing area
%           AR   = aspect ratio
%           S    = sweep
%           TA   = taper
%           TH   = thickness
%           F    = fineness ratio / slenderness
%           Cl   = design lift coefficient
%
%           (optional input parameters)
%
%           dT   = ratio of temperature of next iteration to temperature of
%                  previous iteration (cooling schedule)
%                      default is 0.98
%           run  = number of runs of algorithm
%                      default is 5
%           stop = defines stopping temperature as 10^-stop
%                      default is 10
%
%
% example: [bestvals, bestparams] = annealsizingdrag(1,2,3,4,5,6,7,8)
%
%          or
%
%          [bestvals, bestparams, time, iter]...
```

```
%               = annealsizingdrag(1,2,3,4,5,6,7,8,0.95,10,10)



function [bestvals, bestparams, time, iter]...
          = annealsizingdrag(TW, WS, AR, S, TA, TH, F, Cl, dT, run, stop)


if nargin == 8;                          % simple setup, call with just
                                         % parameters as variables:

    stop = 10;                           % set default minimum temp

    run  = 5;                            % run 5 times for default

    dT = 0.99;                           % set default cooling schedule

end

if nargin == 9;                          % call just how many runs:

    stop = 10;                           % set default minimum temp

    run = 5;                            % set default cooling schedule

end

if nargin == 10;                         % call just how many runs and dT:

    stop = 10;                           % set default minimum temp

end

if stop > 12

    stop = 12;                           % cap minimum temperature at ~eps

end



format long                              % make rand specific enough

p = 1000;                                % mesh search space


TWvec = linspace(2.7, 8 , p)';     % define search space here --
WSvec = linspace(20 , 25, p)';     % can be defined to include
ARvec = linspace(5  , 15, p)';     % definite lower or upper bounds
Svec  = linspace(40 , 200,p)';     % default is to set bounds from
TAvec = linspace(TA*0.8,TA*1.2,p)';     % 80% of original values to 120%
THvec = linspace(TH*0.8,TH*1.2,p)';
Fvec  = linspace(F *0.8,F *1.2,p)';
Clvec = linspace(Cl*0.8,Cl*1.2,p)';

A = [TWvec WSvec ARvec Svec TAvec THvec Fvec Clvec];

                                         % make matrix of values
```

```
%
%         [ lTW lWS lAR lS lTA lTH lF lCl ]
%         [  .   .   .   .  .   .   .  .  ]
%  A =    [ bTW bWS bAR bS bTA bTH bF bCl ]
%         [  .   .   .   .  .   .   .  .  ]
%         [ uTW uWS uAR uS uTA uTH uF uCl ] p x 8 (# of variables)
%


bestparams = zeros(run,8);
bestvals   = zeros(run,1);


tic                                     % start timer


for j = 1:run                           % repeat walk run number of times

TWc = 0.5*p;                            % initialize each variable at
WSc = 0.5*p;                            % initial guess
ARc = 0.5*p;
Sc  = 0.5*p;
TAc = 0.5*p;
THc = 0.5*p;
Fc  = 0.5*p;
Clc = 0.5*p;



str1 = 'Trial '; str2 = int2str(j); str3 = ' of '; str4 = int2str(run);
str5 = '. Please wait...';
                                        % write progress bar dialog

str = [str1 str2 str3 str4 str5];       % compile progress bar dialog

waitbar(0,str);                         % display progress bar

maxiter = 2292;                         % how many iterations the program
                                        % at dT = 0.99

T = 1;

iter = 0;


    while T > 10^(-1*stop)

            iter = iter + 1;

            tempmove = ceil(T*500);         % 500 is empirically derived

            move = ceil(tempmove*rand);     % determine walk direction and
            r = rand;                       % magnitude randomly based on
                                            % current temperature

            if r < 1/3;                     % if r small, decrease value
                TWcn = TWc-move;
```

```
elseif r < 2/3                  % if r medium, keep same
    TWcn = TWc;
else                            % if r large, increase value
    TWcn = TWc+move;            % (step down, stay, or step up
end                             %  with equal probability)


if TWcn > p                     % stop from going outside of
    TWcn = p;                   % boundaries
elseif TWcn < 1
    TWcn = 1;
end

TWin = A(TWcn,1);               % define from value matrix


move = ceil(tempmove*rand);     %
r = rand;                       % repeat for other variables

if r < 1/3;
    WScn = WSc-move;
elseif r < 2/3
    WScn = WSc;
else
    WScn = WSc+move;
end


if WScn > p
    WScn = p;
elseif WScn < 1
    WScn = 1;
end

WSin = A(WScn,2);


move = ceil(tempmove*rand);
r = rand;

if r < 1/3;
    ARcn = ARc-move;
elseif r < 2/3
    ARcn = ARc;
else
    ARcn = ARc+move;
end


if ARcn > p
    ARcn = p;
elseif ARcn < 1
    ARcn = 1;
end

ARin = A(ARcn,3);


move = ceil(tempmove*rand);
r = rand;
```

```
if r < 1/3;
    Scn = Sc-move;
elseif r < 2/3
    Scn = Sc;
else
    Scn = Sc+move;
end


if Scn > p
    Scn = p;
elseif Scn < 1
    Scn = 1;
end

Sin = A(Scn,4);


move = ceil(tempmove*rand);
r = rand;

if r < 1/3;
    TAcn = TAc-move;
elseif r < 2/3
    TAcn = TAc;
else
    TAcn = TAc+move;
end


if TAcn > p
    TAcn = p;
elseif TAcn < 1
    TAcn = 1;
end

TAin = A(TAcn,5);


move = ceil(tempmove*rand);
r = rand;

if r < 1/3;
    THcn = THc-move;
elseif r < 2/3
    THcn = THc;
else
    THcn = THc+move;
end


if THcn > p
    THcn = p;
elseif THcn < 1
    THcn = 1;
end

THin = A(THcn,6);
```

```
            move = ceil(tempmove*rand);
            r = rand;

            if r < 1/3;
                Fcn = Fc-move;
            elseif r < 2/3
                Fcn = Fc;
            else
                Fcn = Fc+move;
            end


            if Fcn > p
                Fcn = p;
            elseif Fcn < 1
                Fcn = 1;
            end

            Fin = A(Fcn,7);


            move = ceil(tempmove*rand);
            r = rand;

            if r < 1/3;
                Clcn = Clc-move;
            elseif r < 2/3
                Clcn = Clc;
            else
                Clcn = Clc+move;
            end


            if Clcn > p
                Clcn = p;
            elseif Clcn < 1
                Clcn = 1;
            end

            Clin = A(Clcn,8);



            val = drag(TWin, WSin, ARin, Sin);
                                            % evaluate


            if iter == 1                    % is this the first iteration?
                newbestval = val;           % then use this as a baseline


            elseif val < newbestval         % is this better?


                newbestval = val;           % if so, tell program its new
%best

                TWi = TWin;                 % update best values for
%variables
```

```
                WSi = WSin;
                ARi = ARin;
                Si  = Sin;
                TAi = TAin;
                THi = THin;
                Fi  = Fin;
                Cli = Clin;

                TWc = TWcn;                 % update matrix location for
                WSc = WScn;                 % variables
                ARc = ARcn;
                Sc  = Scn;
                TAc = TAcn;
                THc = THcn;
                Fc  = Fcn;
                Clc = Clcn;

            else

                if rand < exp((val-newbestval)/(-1*T))

                                            % accept worse solution with
                                            % probability exp(-deltaVal/T)
                                            % so that the program may get
                                            % out of local minima

                newbestval = val;           % if so, tell program new best

                TWi = TWin;                 % update best values
                WSi = WSin;
                ARi = ARin;
                Si  = Sin;
                TAi = TAin;
                THi = THin;
                Fi  = Fin;
                Cli = Clin;

                TWc = TWcn;                 % update matrix location for
                WSc = WScn;                 % variables
                ARc = ARcn;
                Sc  = Scn;
                TAc = TAcn;
                THc = THcn;
                Fc  = Fcn;
                Clc = Clcn;

                end

            end

        if (rem(iter,maxiter/10) == 0)      % update progress bar every
            waitbar((iter/maxiter))         % once in a while
        end

        T = dT*T;

    end


bestparams(j,:) = [TWi WSi ARi Si TAi THi Fi Cli];
```

```
                                                % write best values at the end
bestvals(j) = newbestval;

end

format short                                    % display nicely


time = toc;                                     % record time


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

function [DragF] = drag(df,lf,Aw,Sw)

M=0.4;
H=4500;
nPAX=70;

V=130;
g=9.807;
L=0.0065;
Tzero=288.15;
T=Tzero-L*H;
RHOzero=1.225;
RHO=RHOzero*(1-2.2558/10^5*H)^4.25588;
a=20.0468*sqrt(T);
MIU=1.458/10^6*T^1.5/(T+110.4);
NIU=MIU/RHO;
q=0.5*RHO*V^2;
Rek=V/NIU;

%Input data: ATR 72
Cmac=2.2345; %it can be variabile
CV=0.103;
CH=1.17;
ThicknessRatioH=0.09;
ThicknessRatioV=0.12;
LAMBDAh=0.6;
LAMBDAv=0.6;
TAUh=0.7;
TAUv=0.7;
PHImH=7.531;
PHImV=24.607;
PHIh=7;
e=0.75;
PHIv=24;
xt=0.3;
kcabin=1.1;

%Input equations
b=sqrt(Aw*Sw);
Av=1.6;
Ah=0.38*Aw;
Sv=CV*Sw*b/lf/0.5;
Sh=CH*Sw*Cmac/lf/0.5;
bv=sqrt(Av*Sv);
bh=sqrt(Ah*Sh);
% lbug=df*1.7;
Ref=Rek*lf;
```

```
% lheck=df*3.5;
%lcabin=nPAX/nSA*kcabin
%Sf=df*lcabin+df*lbug/2+df*lheck/2

%Drag in the responsability of the cabin
k=1/pi()/Aw/e;

%Zero lift drag fuselage
FFfus=1+df^3*60/lf^3+lf/df/400;
SwetF=pi()*df*lf*((1-df*2/lf)^(2/3))*(1+df^2/lf^2);
Cffus=0.455/log10(Ref)^2.58/(1+0.144*M^2)^0.65;
Cd0fus=Cffus*FFfus*SwetF/Sw;
D0fus=q*Sw*Cd0fus;

%Zero lift drag horizontal empenage
Reh=Rek*bh;
Cfhorlaminar=1.328/sqrt(Reh);
Cfhorturbulent=0.455/(log10(Reh))^2.58/(1+0.144*M^2)^0.65;
Cfhor=20/100*Cfhorlaminar+80/100*Cfhorturbulent;
FFhor=(1+0.6/xt*ThicknessRatioH+100*ThicknessRatioH^4)*1.34*M^0.18*cos(PHIm
H*pi()/180)^0.28;
Qhor=1.04;
SwetH=2*Sh*(1+0.25*ThicknessRatioH*(1+TAUh*LAMBDAh)/(1+LAMBDAh));
Cd0hor=Cfhor*FFhor*Qhor*SwetH/Sw;
D0hor=q*Sw*Cd0hor;

%Zero lift drag vertical empennage;
Rev=Rek*bv;
Cfverlaminar=1.328/sqrt(Rev);
Cfverturbulent=0.455/(log10(Rev))^2.58/(1+0.144*M^2)^0.65;
Cfver=20/100*Cfverlaminar+80/100*Cfverturbulent;
FFver=(1+0.6/xt*ThicknessRatioV+100*ThicknessRatioV^4)*1.34*M^0.18*cos(PHIm
V*pi()/180)^0.28;
Qver=1.04;
SwetV=2*Sv*(1+0.25*ThicknessRatioV*(1+TAUv*LAMBDAv)/(1+LAMBDAv));
Cd0ver=Cfver*FFver*Qver*SwetV/Sw;
D0ver=q*Sw*Cd0ver;

D0=D0fus+D0ver+D0hor;

%Induced drag;
%Fuselage mass;
Vd=(M+0.09)*a;
mf=0.23*sqrt(Vd*lf/4/df)*SwetF^1.2;

%Empenage mass;
mh=Sh*(62*Sh^0.2*Vd/1000/cos(PHIh)-2.5);
mv=Sv*(62*Sv^0.2*Vd/1000/cos(PHIv)-2.5);

%Lift coeficient;
CIF=(mf+mh+mv)*g/q/Sw;
Cdi=k*CIF^2;
Di=q*Sw*Cdi;

% Objective function
DragF=D0+Di;
```

Orthogonal Steepest Descent with Test Function 2 (OSDsizingdrag.m)

```
% OSDsizingdrag.m
%
% Joseph Yang
% Rice University
% j.yang@rice.edu
% for CARISMA at HAW Hamburg
% 20 July 2010
%
%
% This code applies the Orthogonal Steepest Descent method to optimization
% of preliminary sizing of aircraft design parameters.
%
%
% usage:    [newbestval, bestparams, time]...
%               = OSDsizingdrag(TW, WS, AR, S, TA, TH, F, Cl)
%
% where:
%
%           output parameters:
%
%           newbestval = maximum value of optimization function
%                        with each run's value in each row
%           bestparams = best parameters given by optimization function
%                        organized by row into sets of parameters by trial
%
%           (optional output parameters)
%
%           time       = time elapsed to run code
%           iter       = iterations required to obtain answer
%
%
%           input parameters:
%
%           TW  = thrust to weight ratio
%           WS  = ratio of weight to wing area
%           AR  = aspect ratio
%           S   = sweep
%           TA  = taper
%           TH  = thickness
%           F   = fineness ratio / slenderness
%           Cl  = design lift coefficient
%
%
%
% example: [bestvals, bestparams] = OSDsizingdrag(1,2,3,4,5,6,7,8)
%
%          or
%
%          [bestvals, bestparams, time]...
%               = OSDsizingdrag(1,2,3,4,5,6,7,8,10,5)

function [bestval, bestparams, time, iter]...
                   = OSDsizingdrag(TW, WS, AR, S, TA, TH, F, Cl)


tic

iter = 0;                                % initialize iteration counter
```

```
p = 1000;                                   % mesh search space

step = p*0.2;


TWvec = linspace(2.7, 8 , p)';      % define search space here --
WSvec = linspace(20 , 25, p)';      % can be defined to include
ARvec = linspace(5  , 15, p)';      % definite lower or upper bounds
Svec  = linspace(40 , 200,p)';      % default is to set bounds from
TAvec = linspace(TA*0.8,TA*1.2,p)';     % 80% of original values to 120%
THvec = linspace(TH*0.8,TH*1.2,p)';
Fvec  = linspace(F *0.8,F *1.2,p)';
Clvec = linspace(Cl*0.8,Cl*1.2,p)';

A = [TWvec WSvec ARvec Svec TAvec THvec Fvec Clvec];

TWc = 0.5*p;                        % initialize each variable
WSc = 0.5*p;                        % in the middle of search space
ARc = 0.5*p;
Sc  = 0.5*p;
TAc = 0.5*p;
THc = 0.5*p;
Fc  = 0.5*p;
Clc = 0.5*p;

finished = 0;

newbestval = drag(TW,WS,AR,S);
bestval = drag(TW,WS,AR,S);
bestparams = [TW,WS,AR,S,TA,TH,F,Cl];


while finished == 0;                % run until smallest interval reached

changed = 0;

for a = 1:3                                 % loop over every parameter
    apos = (TWc + (a-2)*step);              % determine step size

    if apos > p
        apos = p;
    elseif apos < 1
        apos = 1;
    end

    aval = A(apos,1);

    for b = 1:3

        bpos = (WSc + (b-2)*step);

        if bpos > p
            bpos = p;
        elseif bpos < 1
            bpos = 1;
        end

        bval = A(bpos,2);
```

```
for c = 1:3

    cpos = (ARc + (c-2)*step);

    if cpos > p
        cpos = p;
    elseif cpos < 1
        cpos = 1;
    end

    cval = A(cpos,3);

    for d = 1:3

        dpos = (Sc + (d-2)*step);

        if dpos > p
            dpos = p;
        elseif dpos < 1
            dpos = 1;
        end

        dval = A(dpos,4);

        for e = 1:3

            epos = (TAc + (e-2)*step);

            if epos > p
                epos = p;
            elseif epos < 1
                epos = 1;
            end

            eval = A(epos,5);

            for f = 1:3

                fpos = (THc + (f-2)*step);

                if fpos > p
                    fpos = p;
                elseif fpos < 1
                    fpos = 1;
                end

                fval = A(fpos,6);

                for g = 1:3

                    gpos = (Fc + (g-2)*step);

                    if gpos > p
                        gpos = p;
                    elseif gpos < 1
                        gpos = 1;
                    end

                    gval = A(gpos,7);
```

```
                            for h = 1:3

                                hpos = (Clc + (h-2)*step);

                                if hpos > p
                                    hpos = p;
                                elseif hpos < 1
                                    hpos = 1;
                                end

                                hval = A(hpos,8);

                                % evaluate:

                                bestval = drag(aval,bval,cval,dval);

                                if bestval < newbestval
                                    newbestval = bestval;
                                    bestparams = [aval bval cval dval...
                                        eval fval gval hval];
                                    TWi = apos;
                                    WSi = bpos;      % if better, update
                                    ARi = cpos;
                                    Si  = dpos;
                                    TAi = epos;
                                    THi = fpos;
                                    Fi  = gpos;
                                    Cli = hpos;
                                    changed = 1;
                                end

                                iter = iter + 1;     % count

                            end
                        end
                    end
                end
            end
        end
    end
end

if changed == 1;                                    % if better soln is
                                                    % found, update
TWc = TWi;
WSc = WSi;
ARc = ARi;
Sc  = Si;
TAc = TAi;
THc = THi;
Fc  = Fi;
Clc = Cli;

elseif changed == 0;                                % if not, stop

    if step == 1;                                   % if smallest step
        finished = 1;                               % already taken, stop
        time = toc;
    end
```

```
    step = ceil(step*0.5);                          % otherwise, refine
                                                    % search


end

end

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


function [DragF] = drag(df,lf,Aw,Sw);

M=0.4;
H=4500;
nPAX=70;

V=130;
g=9.807;
L=0.0065;
Tzero=288.15;
T=Tzero-L*H;
RHOzero=1.225;
RHO=RHOzero*(1-2.2558/10^5*H)^4.25588;
a=20.0468*sqrt(T);
MIU=1.458/10^6*T^1.5/(T+110.4);
NIU=MIU/RHO;
q=0.5*RHO*V^2;
Rek=V/NIU;

%Input data: ATR 72
Cmac=2.2345; %it can be variabile
CV=0.103;
CH=1.17;
ThicknessRatioH=0.09;
ThicknessRatioV=0.12;
LAMBDAh=0.6;
LAMBDAv=0.6;
TAUh=0.7;
TAUv=0.7;
PHImH=7.531;
PHImV=24.607;
PHIh=7;
e=0.75;
PHIv=24;
xt=0.3;
kcabin=1.1;

%Input equations
b=sqrt(Aw*Sw);
Av=1.6;
Ah=0.38*Aw;
Sv=CV*Sw*b/lf/0.5;
Sh=CH*Sw*Cmac/lf/0.5;
bv=sqrt(Av*Sv);
bh=sqrt(Ah*Sh);
% lbug=df*1.7;
Ref=Rek*lf;
% lheck=df*3.5;
```

```
%lcabin=nPAX/nSA*kcabin
%Sf=df*lcabin+df*lbug/2+df*lheck/2

%Drag in the responsability of the cabin
k=1/pi()/Aw/e;

%Zero lift drag fuselage
FFfus=1+df^3*60/lf^3+lf/df/400;
SwetF=pi()*df*lf*((1-df*2/lf)^(2/3))*(1+df^2/lf^2);
Cffus=0.455/log10(Ref)^2.58/(1+0.144*M^2)^0.65;
Cd0fus=Cffus*FFfus*SwetF/Sw;
D0fus=q*Sw*Cd0fus;

%Zero lift drag horizontal empenage
Reh=Rek*bh;
Cfhorlaminar=1.328/sqrt(Reh);
Cfhorturbulent=0.455/(log10(Reh))^2.58/(1+0.144*M^2)^0.65;
Cfhor=20/100*Cfhorlaminar+80/100*Cfhorturbulent;
FFhor=(1+0.6/xt*ThicknessRatioH+100*ThicknessRatioH^4)*1.34*M^0.18*cos(PHIm
H*pi()/180)^0.28;
Qhor=1.04;
SwetH=2*Sh*(1+0.25*ThicknessRatioH*(1+TAUh*LAMBDAh)/(1+LAMBDAh));
Cd0hor=Cfhor*FFhor*Qhor*SwetH/Sw;
D0hor=q*Sw*Cd0hor;

%Zero lift drag vertical empennage;
Rev=Rek*bv;
Cfverlaminar=1.328/sqrt(Rev);
Cfverturbulent=0.455/(log10(Rev))^2.58/(1+0.144*M^2)^0.65;
Cfver=20/100*Cfverlaminar+80/100*Cfverturbulent;
FFver=(1+0.6/xt*ThicknessRatioV+100*ThicknessRatioV^4)*1.34*M^0.18*cos(PHIm
V*pi()/180)^0.28;
Qver=1.04;
SwetV=2*Sv*(1+0.25*ThicknessRatioV*(1+TAUv*LAMBDAv)/(1+LAMBDAv));
Cd0ver=Cfver*FFver*Qver*SwetV/Sw;
D0ver=q*Sw*Cd0ver;

D0=D0fus+D0ver+D0hor;

%Induced drag;
%Fuselage mass;
Vd=(M+0.09)*a;
mf=0.23*sqrt(Vd*lf/4/df)*SwetF^1.2;

%Empenage mass;
mh=Sh*(62*Sh^0.2*Vd/1000/cos(PHIh)-2.5);
mv=Sv*(62*Sv^0.2*Vd/1000/cos(PHIv)-2.5);

%Lift coeficient;
CIF=(mf+mh+mv)*g/q/Sw;
Cdi=k*CIF^2;
Di=q*Sw*Cdi;

% Objective function
DragF=D0+Di;
```