# Optimization Applications 5703 Project B:
# *Aircraft-Based Communications Platform*

Charles O'Neill

23 April 2004

# Contents

# 1 Introduction

The objective of this project is to use constrained objective function minimization techniques to maximize the endurance of an unmanned fan-jet powered communications platform aircraft. This problem has three degrees of freedom and four constraints. Cyclic and steepest descent minimization algorithms[1] are used with a penalty function for the objective function and constraints.

# 2 Problem Description

Conceptually, a high endurance aircraft circles over a service area to provide high-bandwidth communication[2]. The objective is to maximize endurance.

This optimization considers three independent variables: wing span (b), wing area (S), and fuel weight (W). Nomenclature is given in Appendix A (p.3).The variables are *strongly* coupled.

## 2.1 Aircraft Physics

One requirement is accurate modeling of aircraft physics. The modeled flight phases are: takeoff and loiter. These flight stages will be simulated as a steady state maneuvers involving a balance of forces (Fig. 2). Sizing and performance estimates come from *Aircraft Design*[3].

## 2.2 Endurance Function

A simplified jet aircraft endurance expression for a constant lift-to-drag-ratio (Appendix B.1, p.3) depends on aircraft efficiency, engine efficiency, and the amount of fuel. The objective function is non-linear and does not directly contain the independent variables!

## 2.3 Constraints

The constraints specify the aircraft's physics. Constraints (Appendix B.2, p.3) are implemented as penalty functions using squares of bracket operators[1].

- A takeoff ground roll under 8000 feet.
- The loiter phase requires maintaining altitude — thus a positive excess-thrust.
- The geometry limits enforce feasible aircraft configurations. The span is constrained to a positive length less than 200 feet. The wing area is constrained to positive values.

# 3 Optimization Algorithms

This project uses a two level approach: inner local iteration, and an outer ever-increasing penalty function iteration. For the inner loop, two methods were selected: a direct cyclic method and an indirect steepest descent method. Appendix E (p.9) validates both.

## 3.1 Unconstrained Conversion

A penalty function approach[1] converted the problem from constrained to unconstrained. Increasing the penalty function's magnitude forces the unconstrained feasible space to approach the constrained feasible space. The conversion is given in Appendix C.1 (p.5) and coded in Appendix F **penaltyfunction.m**.

## 3.2 Cyclic

The direct cyclic method is implemented as described in Appendix C.2.2 and coded in Appendix F **cyclic.m**. At each iteration, a step is made in a coordinate direction. Step size increases or decreases as consecutive successes or failures occur. The routine stops on small changes in position and functional values, or a small gradient.

## 3.3 Steepest Descent

The indirect steepest descent method —with discrete derivatives— is implemented as given Appendix C.2.3 and coded in Appendix F **sd.m**. A Newton-Raphson line search is made along the search direction in the negative gradient direction. The routine stops on small position or functional changes.

# 4 Algorithm Evaluation

Both algorithms found similar optimums near 52 hours (Appendix D, p.6); however, the algorithms exhibited different behavior.

## 4.1 Efficiency

The cyclic method is more efficient than steepest descent for function evaluations by more than 50 times.

Calculating derivatives and line searching is expensive. Cyclic convergence is more uniform and faster (Figs. 4 and 5).

### 4.2 Comparisons

A feasibility map (Fig. 3, p.7) provided significant insight and served as a comparison. The map gives an optimum of $50^+$ hours. The larger, manned Proteus[2] aircraft provides $20^+$ hour communication service with a heavier payload. Global Hawk[4] has a 32 hour endurance. The proposed optimum is feasible.

### 4.3 Problems Encountered

Modeling and constraint problems appeared. Finding a set of models giving a *reasonable* feasible space was initially difficult. Magnitudes complicated the optimization routines—eg. fuel weight is 100 times larger than wingspan!

Numerical problems also appeared. Matlab allowed complex values —resulting from poor optimization routine guesses— to pollute the variables. The penalty function approach was sensitive to initialization and progression. Small penalty magnitudes created unbounded optimums.

## 5 Summary

A constrained aircraft endurance optimization problem was investigated. The direct cyclic method gave more efficient and robust performance than steepest descent.

## References


[1] High, K., "5703 Optimization Applications Notes," Stillwater, OK, Spring 2004.

[2] Scaled Composites, 1624 Flight Line, Mojave, CA, *Proteus Payload Users Guide*, September 2003.

[3] Raymer, D. P., *Aircraft Design: A Conceptual Approach*, AIAA, 3rd ed., 1999.

[4] Northrop Grumman, I., "RQ-4A Global Hawk," www.is.northropgrumman.com, April 2004.

[5] Abbott, I. H. and Von Doenhoff, A. E., *Theory of Wing Sections*, Dover, New York, 1959.

[6] McCormick, B. W., *Aerodynamics, Aeronautics, and Flight Mechanics*, John Wiley & Sons, New York, 2nd ed., 1995.

# APPENDICES

## A Nomenclature

**Aircraft**

| | |
|---|---|
| A | Aspect Ratio |
| b | Wing Span ($ft$) |
| C | Specific Fuel Consumption ($hr^{-1}$) |
| $C_L$ | Lift Coefficient |
| $C_D$ | Drag Coefficient |
| $C_{D_0}$ | Zero Lift Drag Coefficient |
| D | Drag ($lbs$) |
| E | Endurance ($hr$) |
| e | Oswald's Efficiency Factor |
| q | Dynamic Pressure ($lbs$) |
| S | Wing Area ($ft^2$) |
| $S_G$ | Takeoff Ground Roll ($ft$) |
| T | Thrust ($lbs$) |
| T | Drag ($lbs$) |
| V | Flight Velocity ($ft \cdot s^{-1}$) |
| $V_h$ | Climb Rate ($ft \cdot s^{-1}$) |
| W | Fuel Weight ($lbs$) |
| $\theta$ | Pitch Angle (deg) |

**Atmospheric**

| | |
|---|---|
| g | Gravity ($ft \cdot s^{-2}$) |
| $\rho$ | Density ($slug \cdot ft^{-3}$) |
| $\delta$ | Pressure Ratio |

## B Aircraft Discussion

This section discusses the relevant aircraft physics and modeling in more depth. The basic physical geometry is a conventional aircraft powered by a small fan-jet. The aircraft operates at 50,000 feet, well above most civilian air traffic.

The aircraft layout is shown in Figure 1. The independent variables are wingspan, wing area, and fuel weight. The fuselage volume contains the fuselage structure, fuel and the payload. The payload is a 300 pound, $3 \times 3 \times 5$ foot communications package.

Reviewing some aerodynamics shows that increasing wing area allows the aircraft to create more lift force. Increasing Aspect Ratio (A) improves the efficiency of the wing's lift production. Increasing fuselage volume allows for more internal fuel storage. The conceptual constraints are structural weight, takeoff distance, and fuel capacity. Increasing the aircraft's size increases the structural weight nonlinearly and also increases the overall drag coefficient ($C_{D_0}$).

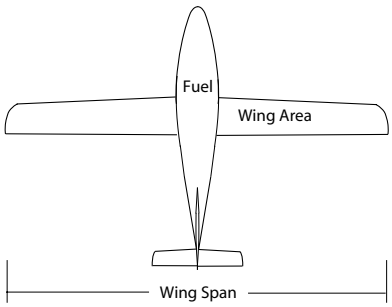


Figure 1: Planform View

The steady state aircraft equations of motion come from a force balance (Fig. 2). Each force is generated as a function of the aircraft's physical geometry and structure.

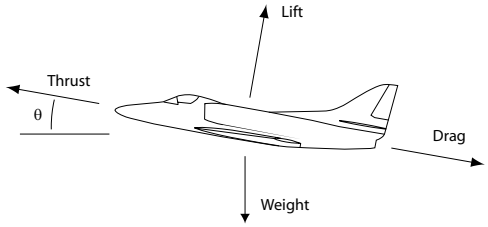


Figure 2: Forces

### B.1 Endurance Function

The objective function is a model of an aircraft's steady state endurance. From Raymer[3], the endurance is maximized at the maximum lift to drag ratio. Endurance as a function of fuel consumption (C) and initial and final weights W is:

$$E = \frac{1}{2}\sqrt{\frac{\pi A\, e}{C_{D_0}}}\left(\frac{1}{C}\right)\ln\left(\frac{W_{initial}}{W_{final}}\right)$$

### B.2 Constraints

This section discusses the details of the objective function constraints. Four constraints are in 3 categories.

#### B.2.1 Takeoff

Takeoff is a thrust and mass dominated maneuver. The ground distance ($S_G$) required when assuming a

constant thrust (Equation 17.101 in Raymer[3]) is:

$$S_G = \frac{1}{2gK_A} \ln\left(\frac{K_T + K_A V_f^2}{K_T}\right)$$

The $K_T$ and $K_A$ terms represent the thrust and aerodynamic forces. $V_f$ is the liftoff velocity, which reduces to:

$$V_f = \sqrt{2W/\rho C_{L_{max}} S}$$

Liftoff velocity function of weight, maximum lift coefficient and wing surface area. For this project, takeoff distance is a constraint. 8000 foot runways are above average length, but are common enough within a given region.

#### B.2.2 Loiter

Loitering requires excess thrust. For this project, the climb rate —positive at the cruise altitude— is governed by the difference between thrust and drag.

$$V_h \propto \left(T - \frac{1}{2}\rho V^2 C_D\, S\right)$$

Thus $V_h$ is required to be positive.

#### B.2.3 Geometry Limits

Geometry constraints were needed to prevent infeasible configurations. The span is limited between zero and 200 feet. The wing area is constrained to be positive.

$$0 < b < 200$$

$$0 < S < \infty$$

### B.3 Aircraft Models

This section discusses the aircraft models used for this project. These models are needed to ensure feasible aircraft performance or configuration estimates.

#### B.3.1 Loiter Flight Condition

A critical assumption for the loiter flight condition regards the aircraft's attitude — pitch angle $\theta$ in Figure 2. Reviewing the endurance expression shows that large lift to drag ratios are desired. From theory reviewed in Raymer[3], the best lift to drag ratio for a finite wing is:

$$\left(\frac{L}{D}\right)_{max} = \frac{1}{2}\sqrt{\frac{\pi A e}{C_{D_0}}}$$

Knowing $C_L$ also allows for determining a nominal flight velocity, which is needed for re-dimensionalizing the drag coefficient $C_D$.

#### B.3.2 Aircraft Weight

One difficult aspect of the aircraft model is the weight estimate. The *Aircraft Design* book by Raymer[3] contains parameterized aircraft component weights. The individual formulas (Equations 15.46 – 15.59 in Raymer[3]) are long, not intuitive nor instructive, and are not included. Weights based on a general aviation aircraft class are estimated for the wing, horizontal, vertical, fuselage, fuel components, landing gear, and engine accessories. The above parameterized weights and aerodynamics appear to have bogus behavior for Aspect Ratios above about 40. The aspect ratio term was adjusted to $(A/cos^2\Lambda)^{0.6} \cdot e^{(A/30)}$ to prevent infeasibly large Aspect Ratio to weight ratios.

#### B.3.3 Drag Model

The drag model considers fuselage and wing drag. Calculating wing drag required estimating an appropriate airfoil section —chosen to be a NACA $65_1412$. *Theory of Wing Sections*[5] provided sectional data. Fuselage drag is calculated from a volume based drag coefficient[6].

#### B.3.4 Engine Performance

The William FJ44 fan-jet engine was selected. The engine provides 1900 pounds of thrust at sea-level and weighs 450 pounds dry. Dimensional analysis[6] suggests that thrust decays with altitude as described by the ambient to sea-level pressure ratio $\delta$.

$$T(h) = \frac{T_{\text{sl}}}{\delta}$$

At 50,000 feet, the thrust decays from 1900 pounds to 217 pounds.

## C Algorithm Discussion

This section discusses application and initialization of the optimization algorithms used for this project. This project concerns a multi-variable, constrained, non-linear objective function.

Two basic optimization methodologies were used: cyclic and steepest descent. Constraint functions were satisfied by adding a penalty function. The Matlab functions and scripts are given in Appendix F.

## C.1 Unconstrained Conversion

The constrained problem was converted to an unconstrained problem with the penalty function method. The objective function has a sign switch to convert from a maximization to a canonical minimization form. The unconstrained objective function becomes:

$$\begin{aligned} P(b,S,W,p) &= \underbrace{-E}_{\text{endurance}} + p \cdot \underbrace{\langle 8000 - S_G \rangle^2}_{\text{takeoff}} \\ &+ p \cdot \underbrace{\langle T - D \rangle^2}_{\text{loiter}} + p \cdot \underbrace{\langle 200 - b \rangle^2}_{\text{wing span}} \\ &+ \underbrace{p \cdot \langle\ S\ \rangle^2}_{\text{wing area}} \end{aligned}$$

As the optimization progresses, the penalty term, p, increases in value. This form is sufficient for implementation in an unconstrained optimization routine.

## C.2 Algorithm Flow Chart

This section details the algorithm flows for the constrained optimization using penalty functions with either cyclic or steepest descent algorithms.

### C.2.1 Outer "Penalty Function" Algorithm

The outer penalty function[1] loop provides the algorithm to satisfy the constraints. The penalty function approach operates by successively increasing a penalty term in the objective function as the optimization progresses. Increasing the penalty term forces the optimization space to approach the constrained feasible space. Overall, this algorithm is easy to code; successively increase a penalty term and recalculate an optimal point. The flow is:

1. Set an initial penalty magnitude $p$ and final objective $p_{max}$
2. Optimize the objective function (cyclic or steepest descent)
3. Increase the penalty magnitude $p = 2 * p$
4. Write out current best values
5. If $p < p_{max}$ then goto 2
6. STOP

The Matlab program is given in Appendix F **endure.m** (p.10).

### C.2.2 Cyclic

The cyclic method is a direct minimization algorithm that operates by successively moving a discrete step *down-hill* in each coordinate direction. The discrete step size depends on the previous successes or failures in the stepping history. Overall, the cyclic method is intuitive, simple to code, and not especially exciting as far as mathematical complexity. The average number of function evaluations per step is 1.5 —50% chance of a succesful single positive evaluation; 50% chance of a failure in the positive direction with a subsequent negative direction evaluation. The flow of the cyclic method is:

1. Set a coordinate search direction
2. Evaluate function f(x) at a positive step $+\alpha$.
3. If the function is smaller, update the current best point. If the function is larger, search in the negative step direction $-\alpha$. If the negative direction is also larger, reduce the step size $\alpha = 0.618\alpha$.
4. If the step direction correlates with previous successful step directions, increase the step size $\alpha = \pi\alpha$.
5. If any two of the following criteria are met, then STOP.
$$\alpha < 10^{-4}$$
$$(x_k - x_{k-1}) < 10^{-4}$$
$$(f_k - f_{k-1}) < 10^{-4}$$
6. If $\nabla f < 1E4$ then STOP
7. Goto 1, unless iterations = 300
8. STOP

The Matlab program is given in Appendix F **cyclic.m** (p.11).

### C.2.3 Steepest Descent

The steepest descent method is an indirect minimization algorithm that operates by successively moving in the negative gradient direction. Thus the update at step k is:

$$x_{k+1} = x_k - \alpha\ \nabla f(x_k)$$

The method requires the gradient $\nabla f(x_k)$ at each point $x$. For this project, the gradient is determined

numerically from a 2 point centered difference expression where $\phi_i$ is evaluated at each coordinate direction.

$$\nabla f(x)_i = \frac{f(x + \Delta \cdot \phi_i) - f(x - \Delta \cdot \phi_i)}{2\Delta}$$

For this problem, $\Delta = 10^{-4}$. Evaluating the gradient for N dimensions with this 2 point expression requires 2N evaluations. The flow of the steepest descent method with a Newton Raphson line search is[1]:

1. Evaluate Gradient $\nabla f(x_k)$
2. Set Search direction $s_k = -\nabla f(x_k)$
3. Determine Newton Raphson components $f'$ and $f''$ in the search direction.
4. If $f'' \neq 0$ Update the position by:
$$x_{k+1} = x_k - \frac{f'}{f''} \cdot s_k$$
5. If the update is not an improvement, move a small length along the search direction[1]
6. If any of the following criteria are met, then STOP.
$$(x_k - x_{k-1}) < 10^{-4}$$
$$(f_k - f_{k-1}) < 10^{-4}$$
$$|\nabla f(x_k)| < 10^{-4}$$
7. Goto 1 unless iterations is greater than 200
8. STOP

The Matlab program is given in Appendix F **sd.m** (p.13).

# D Results

The cyclic method found an optimum of 52.4 hours with 737 function evaluations. The steepest descent method found 52.0 hours with 39672 function evaluations! The final cyclic results are tabulated below for a starting vector of $b = 60, S = 200, W = 2000$

```
Evaluations 737
Endurance 52.37[hr]
Wingspan 51.82[ft]
Area 80.83[ft^2]
Fuel 5610.4[lbs]
Empty Weight 3812.99[lbs]
Takeoff 4922.68[ft]
Aspect Ratio 33.2
Drag 218.71[lbs]
Thrust 218.71[lbs]
Fuselage Volume 244.65[ft^3]
```

The steepest descent results are tabulated below.

```
Evaluations 39672
Endurance 52.05[hr]
Wingspan 55.89[ft]
Area 98.18[ft^2]
Fuel 5539.9[lbs]
Empty Weight 3990.62[lbs]
Takeoff 4141.90[ft]
Aspect Ratio 31.8
Drag 218.74[lbs]
Thrust 218.71[lbs]
Fuselage Volume 243.25[ft^3]
```

Further investigation seems to indicate that the maximum endurance region lies within a flat spot in the feasible space. Small changes in wing span and area are being traded off. However, both methods are finding similar endurance, drag, fuselage volume and aspect ratio. The difference appears to be how much runway the takeoff requires. Thus a smaller wing needs more runway. Further minimization would need to consider multiple objectives such as maintainability or hanger space.

The real surprise is that the cyclic method uses 50 times fewer functional evaluations for essentially the same optimal point! A simple optimization routine appears to be advantageous.

## D.1 Feasible Space

The feasible space of this optimization problem is not trivial. To understand the problem in more detail, an endurance map (Fig. 3) was created. Figure 3 shows cross sections of the feasible configurations and endurance. Endurance is mapped low to high from blue to red. Admittedly this is a brute force approach, but it gave more insight than all of the optimization routines combined![2] The map gives a best endurance of 51 hours with a wing span of 60 feet, a wing area of 114 square feet, and a fuel weight of 5421 pounds. The map also shows the insensitive optimum. Interestingly, the feasible space has a needle pointed valley for small wing spans. The map verifies the optimums predicted by the cyclic and steepest descent methods.

[1] Warning! This is not always a good correction when Newton-Raphson fails.

[2] The map is computationally expensive though.

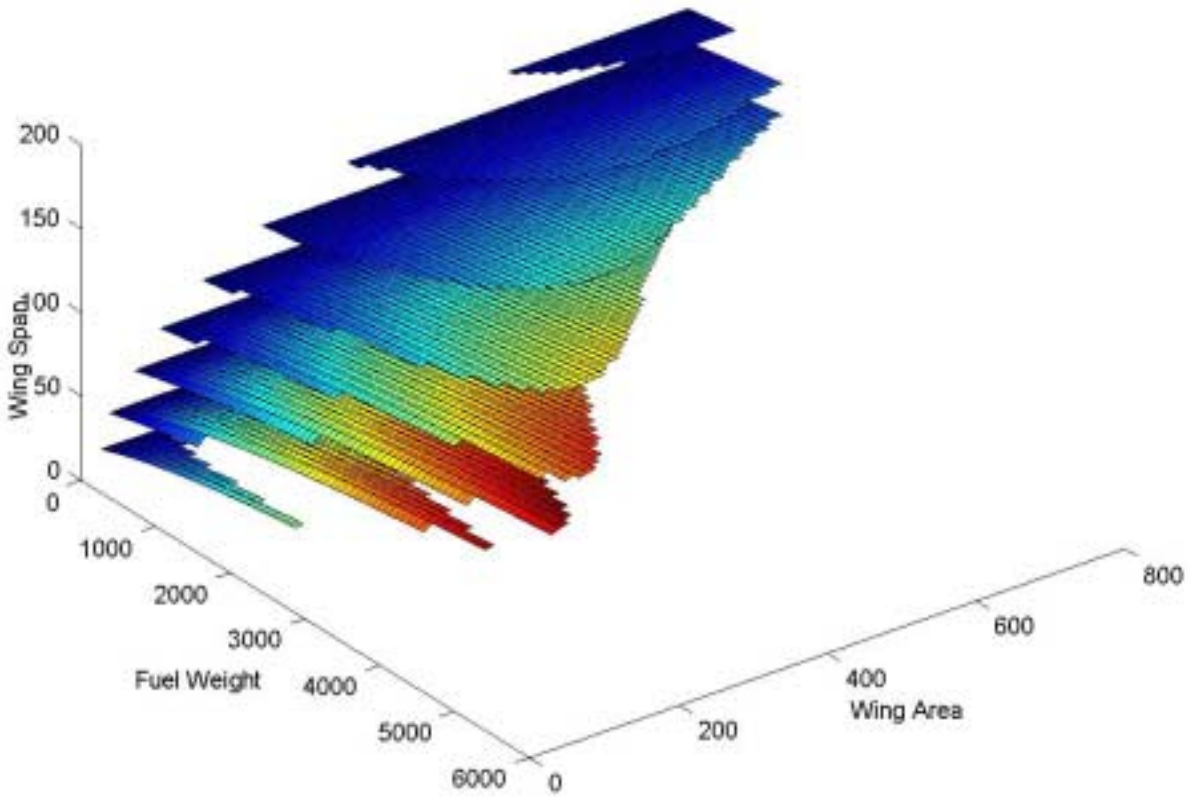


Figure 3: Feasible Space

## D.2 Outer Loop Convergence

The outer loop convergence shows an interesting comparison between the cyclic and steepest descent method. Figure 4 shows the outer convergence of the cyclic method with respect to the three independent variables for three different starting points: a, b, and c:

$$x_a = [20, 40, 300]^T$$

$$x_b = [60, 200, 2000]^T$$

$$x_c = [160, 600, 9000]^T$$

In figure 4, the penaly function effect is apparent from the monotonic convergence to a "steady state" value. Remembering that the constraints enforce physics explains why the values approach from $\infty$. All three cases, a,b, and c, are coincident. The cyclic method converges *fast*! Past the first outer loop, even the initial step differences are gone. Unfortunately, this also explains why the cyclic method prefers a large initial penalty function.

Figure 5 shows the corresponding convergence trace for the steepest descent method. Convergence is not as uniform. A small optimum point difference occurs even with large penalty functions. This appears to be a byproduct of the weak optimum point; small changes in configuration are relatively insensitive to the endurance. This seems to indicate that the steepest descent method as developed for this project is not as efficient as possible. Further study should be done to determine a better local convergence inside the **sd.m** code. Newton-Raphson often fails, so

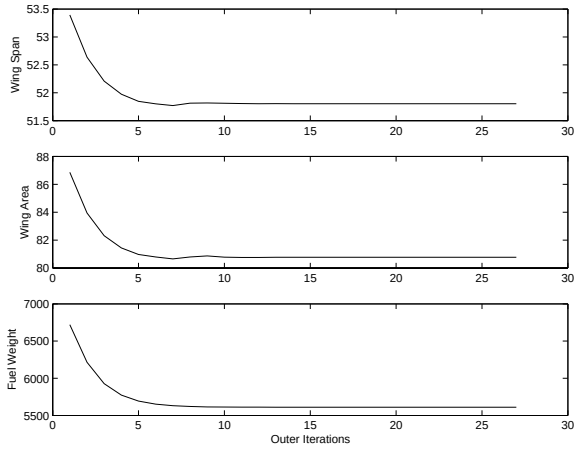


Figure 4: Cyclic Variables' Convergence

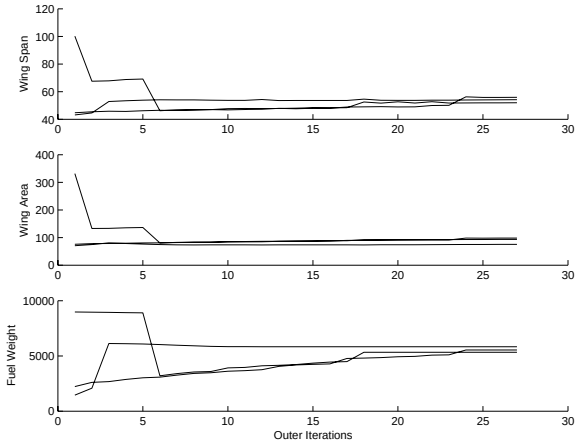


Figure 5: Steepest Descent Variables' Convergence

the alternative discrete step approach is used. After reviewing the code again, the Newton-Raphson only successfully updates within a restricted subset of the feasible space. The Newton-Raphson portion also fails near the minimum, as expected because $f''$ is near zero. This is not good.

## D.3 Comparison

The Proteus aircraft from Scaled Composites allows for a comparison. The Proteus used two William engines yielding significantly higher weight capacity (1500 lbs). The Proteus manual shows 20+ hours endurance at 45000 feet with two engines and 1500 pound payload. This project's optimized aircraft shows similar behavior, but with a much better fuel fraction advantage and a much lighter payload. Longer endurances for specialized aircraft with tiny payloads are possible; the Voyager aircraft —also a Scaled Composites creation— flew around the world without refueling.

This project's aircraft is similar to two other unmanned aircraft, the Global Hawk and the Predator. The Global Hawk operates at 65,000 feet and has a 32 hour endurance[4]. At a lower altitude, the propellor driven Predator operates for 40 hours[3].

# E Hand Calculations

The purpose of this section is to validate the minimization routines on simple, known functions.

## E.1 Polynomial Objective Function Test

Minimization routine validation is necessary. The solution was to create a polynomial test function with a known minimum points. The function is:

$$f(x) = (x_1 - 100)^2 + (x_2 - 600)^4 + (x_3 - 5000)^8$$

which has a minimum of $x = (100, 600, 5000)^T$. The following tables show the cyclic and steepest descent minimization for this polynomial test function. Table 1 for the cyclic method and Table 2 for the steepest descent method show the convergence iterations for the polynomial test function. The direct method converges with fewer function evaluations (72) when compared to the indirect method (703). Both methods stopped on small changes in the functional values.

**Direct**

| Iteration | Evals | $x_1$ | $x_2$ | $x_3$ | $f(x)$ |
|---|---|---|---|---|---|
| 1 | 2 | 0.000000 | 0.000000 | 100.000000 | 3.3 E29 |
| 2 | 3 | 0.000000 | 0.000000 | 414.159265 | 1.9 E29 |
| 3 | 4 | 0.000000 | 0.000000 | 1401.119705 | 2.8 E28 |
| ⋮ | | | | | ⋮ |
| 149 | 40 | 101.703822 | 599.357104 | 4995.431501 | 1.8 E5 |
| 150 | 40 | 101.703822 | 599.357104 | 4995.431501 | 1.8 E5 |
| 151 | 40 | 101.703822 | 599.357104 | 4995.431501 | 1.8 E5 |
| ⋮ | | | | | ⋮ |
| 298 | 72 | 100.005917 | 599.997257 | 4999.998649 | 3.501305 E-5 |
| 299 | 72 | 100.005917 | 599.997257 | 4999.998649 | 3.501305 E-5 |
| 300 | 72 | **100.005917** | **599.997257** | **4999.998649** | 3.501305 E-5 |

Table 1: Polynomial Test with the Direct Cyclic routine

**Indirect**

| Iteration | Evals | $x_1$ | $x_2$ | $x_3$ | $f(x)$ |
|---|---|---|---|---|---|
| 1 | 10 | 0.000000 | 0.000000 | 728.015099 | 1.10 E29 |
| 2 | 19 | 0.000000 | 0.000000 | 1373.263668 | 2.99 E28 |
| 3 | 28 | 0.000000 | 0.000000 | 1898.162997 | 8.56 E27 |
| ⋮ | | | | | ⋮ |
| 37 | 334 | 0.000000 | 0.549403 | 4983.762456 | 1.33 E11 |
| 38 | 343 | 0.000000 | 1.498313 | 4986.384482 | 1.29 E11 |
| 39 | 352 | 0.000002 | 7.461996 | 4991.210486 | 1.23 E11 |
| ⋮ | | | | | ⋮ |
| 76 | 685 | 99.999992 | 600.026170 | 4999.810457 | 2.13 E-6 |
| 77 | 694 | 100.000288 | 600.024818 | 4999.811783 | 2.03 E-6 |
| 78 | 703 | **99.999993** | **600.024787** | **4999.811817** | 1.95 E-6 |

Table 2: Polynomial Test with the Indirect Steepest Descent routine

# F Programs

This project used the Matlab codes given below. The codes are ordered in an approximate execution order. Execution begins by starting the **endure.m** script.

## F.1 Optimization Routine: endure.m

```
% 5703 Project
% Charles O'Neill
% The independent variables are
%   b --- wing span
%   S --- Wing Area
%   V --- Fuselage Volume
clear all;
global penalty; global fevals; global rho; global SFC;

% Initialize
fevals=0;  Pprev=1E99; direct=0;

% Tolerence and Iteration Control
tol=1E-4;
penalty=100;
penalty_max=1E10;
imax=ceil(log(penalty_max/penalty)/log(2));

rho=0.3639E-3; % 50000 ft
SFC=0.48;

x0=[60;200;2000];

b=x0(1);
S=x0(2);
Wfuel=x0(3);

for iter=1:imax
    if(direct)
        xstar=cyclic(x0);
    else
        xstar=sd(x0);
    end
    b=xstar(1); S=xstar(2);       Wfuel=xstar(3);
    P=penaltyfunction(xstar);
    x0=xstar;
    penalty=penalty*2;

    % External output
    fprintf(1,'\n %4.0f & %4.0f & %10.6f &%10.6f  &%10.6f  &%10.6f  ',iter,
        fevals,xstar(1),xstar(2),xstar(3),P)
end

% Write out final optimized parameters
```

```
[fb,thrust]=fuelburn(b,S,Wfuel);
E=endurance(b,S,Wfuel);
SG=takeoff(b,S,Wfuel);
WEIGHT=weight(b,S,Wfuel);
FUEL=Wfuel;
VOLUME=volume(b,S,Wfuel);
DRAG= 1/2*rho*velocity(b,S,Wfuel)^2*S*(drag(b,S,Wfuel)+sqrt(drag(b,S,Wfuel))*(
    pi*b^2/S*0.8)^(-2/3));
AR=b^2/S;
fprintf(1,'\nEndurance %5.2f[hr]  Wingspan %5.2f[ft]  Area %5.2f[ft^2]  Fuel
    %6.1f[lbs]',E,b,S,Wfuel);
fprintf(1,'\nEmpty Weight %5.2f[lbs]    Takeoff %5.2f[ft]    Aspect Ratio %3.1f
    ',WEIGHT,SG,AR);
fprintf(1,'\nDrag %5.2f[lbs]    Thrust %5.2f[lbs]    Fuselage Volume %5.2f[ft^3]
    ',DRAG,thrust,VOLUME);
```

## F.2 Cyclic Direct Routine: cyclic.m

```
% Cyclic
function xstar=cyclic(xm)
func=@penaltyfunction;
global fevals;

% Initial Starting point
dimensions=length(xm);
index=dimensions;
converged=zeros(dimensions,1);
alpha=ones(dimensions,1);
psteps=0; nsteps=0;

% Iterations and Stopping
imax=300;
der_tol=1E-4;
der_delta=1E-4;
x_tol=1E-4;
fxm=feval(func,xm);      fevals=fevals+1;
xpast=xm*1E99; fxpast=xpast;
xpast2=xm*1E99; fxpast2=xpast2;

% Iteration
for iter=1:imax

    %cycle direction
    if(index>dimensions)
        index=1;
    end
    phi=zeros(dimensions,1);
    phi(index)=1;

    %Step and test in cycle direction
    if(feval(func,xm+alpha(index)*phi)<fxm)% Positive move
```

```
34          fevals=fevals+1;
35          xm=xm+alpha(index)*phi;
36          xpast2=xpast;xpast=xm;
37          psteps=psteps+1; nsteps=0;
38      elseif(feval(func,xm-alpha(index)*phi)<fxm)% Negative move
39          fevals=fevals+1;
40          xm=xm-alpha(index)*phi;
41          xpast2=xpast;xpast=xm;
42          nsteps=nsteps+1; psteps=0;
43      else % no move
44          alpha(index)=alpha(index)*0.618;
45          nsteps=0; psteps=0;
46          index=index+1; % Step to next direction
47      end
48
49      %Check convergance and adjust alpha if necessary
50      if((psteps>=1) || (nsteps>=1))
51              alpha(index)=alpha(index)*pi;
52              nsteps=0;
53              psteps=0;
54      end
55
56      % Stopping Criteria
57          %% small x change
58          if(norm(xm-xpast2)<x_tol)
59              converged(1)=1;
60          else
61              converged(1)=0;
62          end
63          %% small alpha
64          if(norm(alpha)<x_tol)
65              converged(2)=1;
66          else
67              converged(2)=0;
68          end
69          %% small f change
70          if(norm(fxm-fxpast2)<x_tol)
71              converged(3)=1;
72          else
73              converged(3)=0;
74          end
75          %% If all are converged, then stop
76          if(norm(converged)^2>dimensions-1)
77              break;
78          end
79
80           %% small jacobian
81          if(norm(converged)^2==2) %check jacobian if mostly converged
82              for i=1:dimensions
83                  phi=zeros(dimensions,1);
84                  phi(i)=1;
```

```
                    fd(i)=(feval(func,xm+der_delta*phi)-feval(func,xm-der_delta*
                        phi))/2/der_delta;
                    fevals=fevals+2;
                end
                if(sqrt(fd*fd')<der_tol)
                    break;
                end
            end
        fxpast2=fxpast; fxpast=fxm;
        fxm=feval(func,xm);

%       % Internal output
%       fprintf(1,'\n %4.0f & %4.0f & %10.6f &%10.6f  &%10.6f  &%10.6f \\\\',
    iter,fevals,xm(1),xm(2),xm(3),fxm)
end
xstar=xm;
return
```

### F.3 Steepest Descent Indirect Routine: sd.m

```
% Steepest Descent
function xstar=sd(xm)
func=@penaltyfunction;
global fevals;

% Initialize
delta=1E-4;
tol=1E-4; %Stopping tolerences
N=length(xm); % Dimensions
imax=200; %Maximum iterations
maxstep=20;
f_past=1E99;
xp=xm; xp_past=1E99;
fxp_past=1E99; fxp=feval(func,xp); fevals=fevals+1;
xpast=xm*1E99; fxpast=xpast;
xpast2=xm*1E99; fxpast2=xpast2;

% Iterate
for iter=1:imax
    %Gradient
    for i=1:N
        phi=zeros(N,1);
        phi(i)=1;
        gradient(i)=1/(2*delta)*( feval(func,xp+phi*delta)-feval(func,xp-phi*
            delta) ); fevals=fevals+2;
    end

    % Stopping Condition
    if((norm(gradient)<tol))
        break;
    end
```

```
31
32      % Search Direction
33      sk=-gradient'/norm(gradient);
34
35      % Derivative
36      fxp_p=feval(func,xp+delta*sk); fevals=fevals+1;
37      fxp_m=feval(func,xp-delta*sk); fevals=fevals+1;
38      fd=(fxp_p-fxp_m)/(2*delta);
39
40      % Newton Raphson Minimization
41      fdd=(fxp_p-2*fxp+fxp_m)/(delta^2);
42      if(fdd<=0) % near a problem since fdd is almost negative
43          xp_kp=xp+sk;
44      else
45          xp_kp=xp-fd/fdd*sk;
46      end
47      fxp_kp=feval(func,xp_kp); fevals=fevals+1;
48
49      % Check proposed point
50      if(fxp_kp<fxp_past) % NR improved guess
51          xp=xp_kp;
52          fxp=fxp_kp;
53      else
54          alpha=1;
55          fprintf(1,'.');
56          xp=xp+sk;
57      end
58      xp_past=xp;
59      fxp_past=fxp;
60
61      % Stopping Criteria
62      %% small x change
63      if(norm(xp-xpast2)<tol)
64          break;
65      end
66      %% small f change
67      if(norm(fxp-fxpast2)<tol)
68          break;
69      end
70      fxpast2=fxpast; fxpast=fxp;
71      xpast2=xpast; xpast=xp;
72
73      %       % Internal output
74      %     fprintf(1,'\n %4.0f & %4.0f & %10.6f &%10.6f  &%10.6f  &%10.6f \\\\',
            iter,fevals,xp(1),xp(2),xp(3),fxp)
75  end
76  xstar=xp;
77  return
```

## F.4 Penalty Function: penaltyfunction.m

```
% Objective Function with Penalties
function P=penaltyfunction(xm)
global penalty; global rho;
e=0.80;
b=xm(1);
S=xm(2);
Wfuel=xm(3);

%Constraints
[fb,thrust]=fuelburn(b,S,Wfuel);
Q=1/2*rho*velocity(b,S,Wfuel)^2;
takeoff_max=8000;
span_max=200;

% Takeoff Length
h1=min(0, (takeoff_max-takeoff(b,S,Wfuel))/takeoff_max)^2;

% Wing Span Limit
h2=min([0, (span_max-b)/span_max, b])^2;

% At-Altitude Minimum Climb Rate
CD=drag(b,S,Wfuel)+sqrt(drag(b,S,Wfuel))*(pi*b^2/S*e)^(-2/3);
DRAG=Q*S*CD;
ExcessThrust=thrust-DRAG;
h3=min(0,ExcessThrust/thrust)^2;

% Wing Area Limit
h4=min([0,S])^2;

%Objective Function
f=endurance(b,S,Wfuel);
P=-f+(h1+h2+h3+h4)*penalty;

return
```

### F.5 Endurance: endurance.m

```
% Endurance
function hours=endurance(b,S,Wfuel)
global SFC
AR=b^2/S; %Aspect Ratio
e=0.80; %Oswalds Wing Efficiency Factor
CLCD_best=sqrt(max(0,pi*AR*e/drag(b,S,Wfuel))); %Best Lift to Drag Ratio
Fuel_fraction=(weight(b,S,Wfuel)+Wfuel)/(weight(b,S,Wfuel)+100);
hours= 1/2 * CLCD_best / SFC * max(0, log(Fuel_fraction));
return
```

### F.6 Weight: weight.m

```
%    weight model weight(b,S,Wfuel)
function weight_total=weight(b,S,Wfuel)

%% Physical Constants
AR= b^2/S; % Aspect ratio = span squared / Area
chord=max(0,S/b); % wing chord
lambda=0.4; % taper ratio
N_z= 8; % ultimate load factor
D=4; %Fuselage diameter
L=20; %Fuselage length
s_f= L*D*pi; %Fuselage wetted area
L_t= 10; % Tail length
tc=0.12; % thickness
W_dg= 2800+Wfuel*0.5; % Design Flight weight
q=200; %Design Dynamic Pressure

% wing weight (general aviation Raymer p476)
wingweight=0.036*S^0.758 * Wfuel^0.0035 * AR^0.6* exp(AR/30) * q^0.006 *
    lambda^0.04 * (100 * tc)^-0.3 * (N_z * W_dg)^0.49;

% Fuselage (general aviation Raymer p476)
fuselageweight=0.052* s_f^1.086 * (N_z*W_dg)^0.177 * L_t^-0.051 * (L/D)
    ^-0.072 * q^0.241;

% Horizontal (general aviation Raymer p476)
SSh_ratio=0.50*chord/L_t;
horzweight=0.016* (N_z * W_dg)^0.414 * q^0.168 * (S*SSh_ratio)^0.896 * (100*tc
    )^-0.12 * (AR)^0.043  * lambda^-0.02;

% Vertical (general aviation Raymer p476)
verticalweight=0.7* horzweight;

% Landing Gear
LGweight=100;

% Powerplant weight
W_en=450; % dry engine weight
W_powerplant=W_en*1.3; %(general aviation Raymer p476)

% Avionics
Avweight=100;

% Comm Payload
W_payload=300;

%%% TOTAL WEIGHT %%%
weight_total=wingweight+fuselageweight+horzweight+verticalweight+LGweight+
    W_powerplant+Avweight+W_payload;

return
```

### F.7 Fuselage Volume: volume.m

```
% Volume
function volume_total=volume(b,S,Wfuel)
%Fuselage Structure
vol_fuse_structure=20*2*2;
%Payload volume
vol_payload=3*3*5;
%Avionics and Flight Systems Volume
vol_avionics=2*2*2;
%Fuel Volumes
vol_fuel=Wfuel/50.25;
%Total Volume
volume_total=vol_payload+vol_avionics+vol_fuse_structure+vol_fuel;
return
```

### F.8 Drag: drag.m

```
%   Drag model
function cdrag=drag(b,S,Wfuel)
% wing drag
Cd_wing=0.011;

% Fuselage Drag
% Drag per volume^(2/3) for a Fuselage/Nacelle
CD_V=0.030; % McCormick p165

%%% TOTAL DRAG %%%
cdrag=Cd_wing+CD_V*volume(b,S,Wfuel)^(2/3)/S;
return
```

### F.9 Fuel Burn: fuelburn.m

```
% Fuelburn
function [lbs,thrust]=fuelburn(b,S,Wfuel)
global SFC
delta=(243.6/2116.2); %50000 feet
thrust=1900*delta;
hours=endurance(b,S,Wfuel);
lbs=SFC*hours*thrust;
return
```

### F.10 Takeoff Distance: takeoff.m

```
% Takeoff Distance
function SG=takeoff(b,S,Wfuel)
% Rolling Resistance
mu=0.04;
% Gravity
```

```
g=32.2;
% Sea Level Density
rho_sl=2.37E-3; % slug / ft^3
% Takeoff Lift Coefficient
CL_to=2.0;
% Thrust at SL
thrust=1900;
% Takeoff Velocity
Vf2=2*(Wfuel+weight(b,S,Wfuel))/rho_sl/CL_to/S;
KT=(thrust/(Wfuel+weight(b,S,Wfuel)))-mu;
KA=-rho_sl*S/2/(weight(b,S,Wfuel)+Wfuel)*drag(b,S,Wfuel);
% Takeoff Distance from Raymer p 565
SG=(2*g*KA)^-1 * log((KT+KA*Vf2)/(KT));
return
```

### F.11 Feasibility Map: mapout.m

```
% Feasibility Map
clear all;
global rho;
global SFC;
global penalty
for b=20:20:200
    %b=100;
    rho=0.3639E-3; % 50000 ft
    SFC=0.48;
    imax=70;
    jmax=70;

    fmax=5500/imax;
    smax=800/jmax;

    penalty=10000000;

    for i=1:imax
        Wfuel(i)=i*fmax;
        for j=1:jmax
            S(j)=j*smax;

            [PP(j,i),h1,h2,h3,h4]=obj([b,S(j),Wfuel(i)]);

            constraints=((h1==0) && (h2==0) && (h3==0) && (h4==0));

            if(constraints)
                E(j,i)=endurance(b,S(j),Wfuel(i));
                if(E(j,i)==0)
                    E(j,i)=NaN;
                end
            else
                E(j,i)=NaN;
            end
```

```
35
36                  if(PP(j,i)>10)
37                      PP(j,i)=NaN;
38                  end
39              end
40          end
41          %subplot(2,1,1)
42          surface(Wfuel,S,E+b,E); hold on;
43          %subplot(2,1,2)
44          %surface(Wfuel,S,PP); hold on;
45          [J,I]=find(E==max(max(E)));
46          fprintf(1,'----------')
47          WF_max=I*fmax
48          S_max=J*smax
49          b
50          E=endurance(b,S_max,WF_max)
51
52  end
```