To the Graduate Council:

I am submitting herewith a thesis written by John K. Gibby entitled "Application of the MCA Method to Determine In-Flight Drag of a Propeller Driven Aircraft." I have examined the final copy of this thesis for form and content and recommend it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Aerospace Engineering.

Ralph D. Kimberlin

R. D. Kimberlin, Major Professor

We have read this thesis
and recommend its acceptance:

William S. Oakley

Charles C. Limbaugh

Accepted for the Council:

Lew Minkel

Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature John Keith Kelly

Date 17 April 1990

APPLICATION OF THE MCA METHOD TO DETERMINE IN-FLIGHT DRAG OF A PROPELLER DRIVEN AIRCRAFT

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

John Keith Gibby

May 1990

ACKNOWLEDGEMENTS

I am deeply thankful to Professor Ralph Kimberlin for his guidance throughout this work. He suggested the study topic and helped greatly with his experience in the flight test arena. I would also like to thank the other committee members, Dr. Charles Limbaugh and Dr. William Baker, Jr. for their time involved in this effort.

I am also very grateful to my supervisor Lt. Col. Robert Chedister who gave me time away from my other duties so that this study could be completed.

Finally, my warmest thanks goes to my wife, Norma, for her support and encouragement during this study.

ABSTRACT

The Mass Consumption Acceleration (MCA) method has been used successfully to determine the in-flight drag of a jet aircraft with fixed area nozzles. The purpose of this study was to apply the method to an aircraft with a reciprocating engine turning a constant speed propeller. The aircraft and engine were modeled with a computer program to generate simulated flight test data. The data were then analyzed with the MCA method. The method is shown to be both theoretically and analytically valid for application to a propeller driven aircraft.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

D	drag (lb)
$\dot{e}$	excess specific energy (ft/sec)
fps	feet per second
g	gravitational constant (32.2 ft/sec^2)
gph	gallons per hour
h	altitude (ft)
H	fuel heating value (BTU/lb)
m	mass (slugs)
$\dot{m}$	fuel flow (slugs/sec)
P_{AUX}	aircraft auxiliary power (hp)
P_{OUT}	effective power output (hp)
P_{REQ}	Power Required (hp)
rpm	revolutions per minute
s	flight path distance (ft)
t	time (sec)
T	thrust (lb)
V	true airspeed (ft/sec)
W	weight (lb)
η	efficiency factor

Subscripts

E	engine parameter
R	reference value
0	initial point
*	solution reference point

CHAPTER 1

INTRODUCTION

At some point during the flight test of a new aircraft or aircraft modification it is necessary to measure the in-flight thrust and drag. In many cases, such as transports, where the drag has a significant effect on the range, it must be known to a high degree of accuracy. In other examples such as a general aviation aircraft the drag, still of importance, can be determined with less exactness.

Typically, to find drag one must know the installed thrust of the engine(s). The thrust determination is based on manufacturer's ground test data for a calibrated engine. While this method is accurate it can be quite expensive. Several variations of basic methods are described in [1][1] and [2].

Casetti [3] first introduced the basic Mass Consumption Acceleration Method. The method was developed to the present form by Rosenberg [4]. The MCA method provides a quick, inexpensive means to determine in-flight thrust and hence, drag. To date this method has only been applied to jet aircraft.

---

[1]Numbers in brackets refer to similarly numbered sources in the list of references.

This study will focus on applying the method to an aircraft with a single reciprocating engine and a constant speed propeller. The analysis was done using a digital computer program to model the aircraft and engine. Data from this program were then reduced using the MCA method.

CHAPTER 2

REVIEW OF THE MCA METHOD

This method is derived from the Law of Conservation of Energy. That is the thermal energy, contained in the fuel, is converted into mechanical energy. A more detailed derivation of the MCA method can be found in [4].

First, the power expended to overcome external forces on the aircraft are equated to the change in mechanical energy.

$$(T-D)\frac{ds}{dt} = mg\left[\frac{dh}{dt} + \frac{V}{g}\frac{dV}{dt}\right] \tag{1}$$

The right hand side of the equation divided by the weight is the rate of change of specific energy, or excess specific energy

$$\dot{e} = \frac{V}{g}\frac{dV}{dt} + \frac{dh}{dt} \tag{2}$$

and the work per unit time done along the flight path is

$$mg\dot{e} = V(T-D) \tag{3}$$

The propulsive force (T) is directly related to throttle setting and the drag force (D) will include trim effects as a function of surface deflection which is dependent on throttle setting. The thermal energy input to the engine is related to the mechanical energy output from the engine by the efficiency factor, η, as follows:

$$\eta = \frac{(TV + P_{AUX})}{\dot{m}H} \tag{4}$$

Note that the auxiliary power term, a fixed power loss for the aircraft accessories, could be considered an offset or bias to the engine output so that the efficiency could also be expressed as a function of thrust and velocity in the numerator. The effect of this will be seen in the final thrust equation.

Now combining Equations (3) and (4)

$$\eta \dot{m}H = W\dot{e} + DV + P_{AUX} \tag{5}$$

To develop the method further the values of weight, drag, and airspeed are held constant. Equation (5) is now differentiated with respect to $\dot{m}$ to give

$$\left(\frac{d\eta}{d\dot{m}}\right)\dot{m} + \eta = \frac{W}{H}\frac{d\dot{e}}{d\dot{m}} \tag{6}$$

Now replacing η with Equation (4) yields a relationship for thrust

$$T = \frac{1}{V}\left[W\dot{m}\frac{d\dot{e}}{d\dot{m}} - H\dot{m}^2\frac{d\eta}{d\dot{m}} - P_{AUX}\right] \tag{7}$$

It is now a matter of measuring W, $\dot{m}$, $\dot{e}$, V, and P_{AUX} in flight. Note that if the case of $d\eta/d\dot{m} = 0$ can be found then the thrust can be determined from Equation (7). To further define the operational points where $d\eta/d\dot{m} = 0$ differentiate

Equation (3) while still holding weight, drag, and airspeed constant, so in incremental form

$$\frac{W}{V}\Delta\dot{e}=\Delta T \tag{8}$$

Now designate the reference values of ė and T with an R subscript, where $d\eta/d\dot{m}=0$, and have values for the incremental operating point denoted without index so that

$$T=\frac{W}{V}(\dot{e}-\dot{e}_R)+T_R \tag{9}$$

Equation (9) still presumes that mass, drag, and airspeed are constant. Proceeding, substituting the thrust term from Equation (7) for the reference value in Equation (9) and assuming $d\eta/d\dot{m} = 0$

$$T_R=\frac{W}{V}\dot{m}_R\left(\frac{d\dot{e}}{d\dot{m}}\right)_R-\frac{P_{AUX}}{V} \tag{10}$$

Again using $d\eta/d\dot{m} = 0$ and combining Equations (4), (9), and (10) the efficiency factor in terms of reference values becomes

$$\eta=\left(\frac{W}{\dot{m}H}\right)\left[\dot{e}-\dot{e}_R+\dot{m}_R\left(\frac{d\dot{e}}{d\dot{m}}\right)_R\right] \tag{11}$$

To obtain the data for this procedure, specific operational procedures must be followed. Test runs must be conducted at a constant altitude and throttle setting with the wings level. The test run is initiated just above stall speed

for accelerations and at maximum airspeed for decelerations. During the run the pilot attempts to maintain altitude by trimming the aircraft. Throughout the run, time, airspeed, fuel flow, and altitude are recorded until the airspeed reaches a constant value or a flight safety limit is reached. Once the speed range has been covered at one altitude with several throttle settings, the process is repeated at other altitudes until the entire altitude and airspeed envelope has been investigated.

Once the data are collected corrections for non-standard conditions, atmosphere and weight, are applied using methods described in [5]. Additionally, the calibrated airspeed must be converted to true airspeed. The excess specific energy ($\dot{e}$) is calculated using Equation 2. Values of $\dot{e}$ and $\dot{m}$, at the different throttle settings, are plotted against the true airspeed for a constant altitude, see Figure 1[2]. Next, the data are faired and a cross-plot of $\dot{e}$ versus $\dot{m}$ is created by choosing points at a constant airspeed, see Figures 1 and 2.

For each curve at a constant airspeed families of efficiency curves are generated using Equation 11. First a reference point ($\dot{m}_R$, $\dot{e}_R$) is selected, and the slope ($d\dot{e}/d\dot{m}$) is calculated. Then $\dot{m}$ is varied from the minimum to the maximum fuel flow for the constant airspeed curve. A curve of η versus $\dot{m}$ is created, Figure 3, that has a maximum at $\dot{m}_R$ where $d\eta/d\dot{m} = 0$. Another reference point is selected and the

---

[2]All figures are shown in Appendix 1.

process is repeated, again from the constant airspeed curve. The objective of this effort is to determine the efficiency curve which would pass through $\eta = 0$ where $d\eta/d\dot{m}=0$ also.

Once the proper curve is found the reference points used are then designated $\dot{e}_*$ and $\dot{m}_*$ and the slope becomes $(de/d\dot{m})_*$. Now combining Equations (9) and (10) and substituting $\dot{m}_*$ and $\dot{e}_*$ the thrust can be expressed as

$$T = \frac{W}{V}\left[\dot{e} - \dot{e}_* + \dot{m}_*\left(\frac{d\dot{e}}{d\dot{m}}\right)_*\right] - \frac{P_{AUX}}{V} \qquad (12)$$

Note that the auxiliary power term is simply an offset and that the analysis can be performed by neglecting the term throughout the data reduction and applying it to the thrust prior to further thrust dependent calculations.

Now, for $\eta = 0$ in Equation (11) substitute $\dot{m}_*$, $\dot{e}_*$, and $\dot{e}_0$ at $\dot{m} = 0$ so that

$$\dot{e}_0 - \dot{e}_* + \dot{m}_*\left(\frac{d\dot{e}}{d\dot{m}}\right)_* = 0 \qquad (13)$$

which allows Equation (12) to be expressed as

$$T = \frac{W}{V}[\dot{e} - \dot{e}_0] - \frac{P_{AUX}}{V} \qquad (14)$$

Recalling Equation (3) and substituting $\dot{e}_*$ and the thrust value, T_*, gives

$$D = T_* - \frac{W}{V}\dot{e}_* \tag{15}$$

This is the drag for the airspeed and altitude of interest. To generate a drag polar the analysis is repeated for other constant airspeed curves across the speed range of the data.

Once again the key to gaining the proper solution lies in finding the efficiency curve that passes through $\eta = 0$, where $d\eta/d\dot{m}=0$, at a zero fuel flow. This solution becomes easier to determine if the idle fuel flow lies close to zero so that one does not have to extrapolate too far. The zero fuel flow point is only obtainable with a windmilling engine which is hardly advisable from a safety aspect on a high performance single engine aircraft. However, if the aircraft has more than one engine the efficiency curve can be taken closer to zero fuel flow by using one engine windmilling and having the other engine(s) at idle. As noted in [4] this procedure has been successfully applied to a twin engine jet aircraft.

Limitations of the MCA method on a single engine jet aircraft were found by the USAF Test Pilots School [6] when the procedure was applied to measure the in-flight thrust of an A-7D. The engineers found it difficult to accurately predict drag because the efficiency curves could not be realistically extrapolated to zero.

To date the method has only been applied to jet aircraft with fixed area nozzles. In an effort to further utilize this method this study will apply it to an aircraft with a normally

aspirated reciprocating engine which uses a constant speed propeller. This aircraft type was selected as an initial starting point for propeller driven aircraft because the fuel flow can be modeled at a constant rpm which reduces the number of variables to those that the MCA method uses.

To demonstrate that the method is potentially applicable to this aircraft model, a review of the derivation is needed. The fundamental Equations (1) - (3) hold true because they represent the effects of external forces on the change in mechanical energy. This relationship is applicable to any aircraft which has a fixed line of thrust that can be reduced to a function of fuel flow, altitude, and airspeed [4].

The other fundamental relationship between thermal and mechanical energy is the efficiency factor. The efficiency of the overall aircraft, [7] and [8], can be described as follows

$$\eta = \eta_P \times \eta_E \tag{16}$$

For the propeller efficiency

$$\eta_P = \frac{P_{OUT}}{P_E} \tag{17}$$

For the engine efficiency the auxiliary power increment is considered as an offset to the engine power as previously discussed such that

$$\eta_E = \frac{P_E}{\dot{m}H} \tag{18}$$

Substituting Equations (17) and (18) and P_{OUT}= TV into Equation (16)

$$\eta = \frac{TV}{\dot{m}H} \tag{19}$$

This is the same term as Equation (4) without the auxiliary power represented. As previously shown, Equation (12), the thrust equation can be derived with this increment determined prior to the drag calculations. Based on this analysis the MCA method appears to be applicable to this aircraft type.

CHAPTER 3

DESCRIPTION OF THE AIRCRAFT AND ENGINE MODEL

In order to fully evaluate the application of the MCA method to an aircraft with a reciprocating engine a large data base is necessary. While use of an actual aircraft would be most representative, the use of an ideal aircraft, or math model, would provide the best means to demonstrate the theory. Therefore, a computer model of an aircraft was used to generate the required data base of simulated flight data.

For the purpose of this study certain simplifying assumptions were made. First, no power load, P_{AUX}, due to avionics or other systems was modeled. Second, the actual weight was held constant because the percentage mass change for this airplane is very small. Third, throttle settings are step inputs rather than a ramp model. Finally, the test run is terminated when the velocity change is less than 0.5 fps.

The parameters for this model were obtained from [9]. The aircraft modeled is a Piper Arrow with retractable gear. The engine model is a 200 horsepower Lycoming O-360-A without turbocharging. The airplane utilizes a constant speed propeller which for this case will run at 2500 rpm. As previously stated no auxiliary power load for avionics will be used because it is a constant value that can be applied later.

The program, MCA1, that models this aircraft is shown in Appendix 2. It is written in standard Pascal [10] except for

the input and output declarations which are for use with the Digital VAX 11/780. The interpolation routine is Neville's algorithm from [11], and the smoothing routine used is a polynomial least squares curve fit adapted from [12].

The program is designed to simulate test runs at different altitudes, however, for this investigation only those runs at sea level, standard day conditions were performed. The program requires an initial starting airspeed, either stall speed or maximum airspeed, and power setting. This is an iterative process where the user estimates an initial power and the program returns the actual power output, which is repeated until the estimated value agrees with the actual power. Next, a throttle setting is entered that will accelerate or decelerate the aircraft. As the run is executed the data are collected in a separate file. The run is terminated when the velocity change becomes negligible. This process is repeated for as many throttle settings as desired.

For the engine model the required information is power output as a function of fuel flow at sea level and constant 2500 rpm. Figure 4 shows the Lycoming engine performance chart for the O-360-A. This chart can be used for all altitudes and nonstandard conditions; however, only the left chart is needed to get standard day, sea level conditions. At 2500 rpm the data, from Figure 4, for brake horsepower were plotted against fuel flow, see Figure 5. To get the lowest fuel flow for 2500 rpm the engine performance was extrapolated

to 5.7 gph. The data in Figure 5 were fit with a second order polynomial. This polynomial is used in the program procedure FUELPOWER. The procedure will return either fuel flow or brake horsepower as required.

Procedure CHANGEPOWER prompts the user for a throttle setting. The acceptable values for acceleration or deceleration are displayed to the user. The setting is a decimal number from zero to one (100% throttle). Once the user enters the value the procedure calls FUELPOWER which returns the brake horsepower.

In order to maintain a constant 2500 rpm, so that the number of variables may be reduced, a constant speed propeller is used. The data for the propeller are shown in Figures 6 and 7. These curves were digitized and are represented in the procedure INITIALIZECONSTANTS. When the power from the engine into the propeller is known the advance ratio and power coefficient are calculated and function BLADEANGLE calculates the propeller blade angle. Once the angle is known the propeller efficiency is determined by the function INTERPOLATE in procedure PROPELLEREFFICIENCY. It was found that interpolation of the propeller charts lead to some discontinuities in the efficiency curve causing small jumps in the acceleration curve which were not realistic. To avoid this problem, an initial sweep of the speed range is done using PROPELLEREFFICIENCY to generate values for each new power setting. These data are then smoothed with a second

order polynomial across the speed range using procedure SMOOTHPROPELLEREFFICIENCY. The coefficients from this equation are then used to generate all values of propeller efficiencies for the power setting.

As the aircraft accelerates it is necessary to know the drag at any given airspeed. The drag model for this aircraft is the power required versus true airspeed in Figure 8. The procedure, THP, uses a second order fit to represent the curve. This procedure is also used later to generate values for comparing the MCA solution to the model. The equation to compute drag is

$$D = 550 \frac{P_{REQ}}{V} \tag{20}$$

The drag is recalculated for each second of elapsed time.

Procedure FLYPOINT simulates the acceleration or deceleration by the following relationship:

$$F = T - D \tag{21}$$

which is the force balance for the aircraft which is also

$$F = \frac{W}{g} \frac{dV}{dt} \tag{22}$$

so, in incremental form

$$\Delta V = g \frac{\Delta t}{W} (T - D) \tag{23}$$

This incremental change in airspeed is added to the true airspeed. This process is continued interatively until the change in airspeed is less than 0.5 fps. At each point $\dot{e}$ is calculated, see Figure 9, and stored in a data file with time, fuel flow, airspeed, and thrust. For this investigation, 54 runs were performed from 47% to 100% throttle in 1% increments.

CHAPTER 4

APPLICATION OF THE MCA METHOD

A second program, MCA2, is used to format the data and is shown in Appendix 2. This program uses data output from MCA1 and puts values in the form required for the MCA method.

Since the MCA method requires values at a constant airspeed it is necessary to cross-plot $\dot{e}$ versus $\dot{m}$ for a given airspeed. Airspeeds from 95 to 205 fps were chosen in 5 fps increments. At each airspeed division, SPEEDTIC, data samples of +/-20 fps were taken, except at endpoints. The data samples $\dot{e}$ and thrust were interpolated with the point of interest as the unknown. The interpolated values for each airspeed are stored in an new array.

The new arrays are indexed with an integer (SPEEDINDEX) representation of the airspeed. The arrays are then sorted in ascending order by fuel flow by procedure SORTDATA. The data now in terms of constant airspeeds are written to a file to be used by the program MCA3.

The program MCA3, see Appendix 2, applies the equations derived earlier to the sorted data to compute thrust and drag. It then compares the solution to the math model values.

First each constant airspeed data set is individually read in, see Figure 10. Because the interpolation in MCA2 introduces some discontinuities the data ($\dot{e}$ and T) are smoothed using a cubic degree polynomial which gave the best

results over other fits. This is especially important for the $\dot{e}$ versus m curves where a local change in slope would propagate through the calculations, see Equation (15), and give incorrect results.

The reference points are chosen at each fuel flow, determined by throttle setting. Using Equation (11) the efficiency curve for this reference point is generated for the minimum to maximum values of fuel flow. This procedure is repeated for each fuel flow reference point so that the power setting range is covered.

Drag is only calculated at the reference points since that is the only place that it has any meaning. Recall that there is only one reference point that will give the correct drag value. To determine if such a point exists all reference points are listed with the associated lift and drag values.

By examining data output in this manner the minimum drag error is easily found. Note that this method does not define a procedure to locate the proper efficiency curve. Instead it shows whether or not a reference point can be found to solve for the thrust and drag, which would validate the method. Data of this type are shown in Tables 1[3], 2, and 3. Figure 11 shows a sample of efficiency curves at 150 fps. The middle curve is the correct solution based on data from Table 2.

To validate the method it is necessary that the thrust and drag both be close to the correct value for each force.

---

[3]All tables appear in Appendix 3.

For example, the method would not be considered valid if the drag value were close but the thrust value was not. For this study an absolute error less than seven percent [4] is adequate to show that the method is valid.

Examining the values in Tables 1 through 3 the minimum values for drag can be located for each reference point. The results from application of the MCA method to the model indicate that there are values for the reference points that give good approximations for both thrust and drag values. Good correlation was found across the speed range as shown in Table 4.

The data, in Tables 1 through 3, also show the effect of fuel flow on the solution. Note that for most cases the data are very sensitive to fuel flow, errors become unacceptable for values greater than 0.10 gph. This result means that actual flight test of the aircraft using this method will require accurate measurements of the data.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

The MCA method which has previously only been applied to jet aircraft was applied to an airplane with a reciprocating engine turning a constant speed propeller. The analysis was done with the aid of a digital computer to model the aircraft, reduce data, and apply the MCA method.

The method, shown to be theoretically possible, proved to give valid results for thrust and drag. Before the method can be applied to an actual aircraft in flight test a procedure to pick the proper efficiency curve to get the correct reference point is required.

In addition, a sensitivity analysis, using an aircraft math model, should be done on all in-flight measured data. Information from such a study should be used to define the instrumentation accuracy for an actual flight test.

Finally, it may be also be possible to apply the method to an aircraft with a fixed pitch propeller. It is recommended that a similar analytical approach using a computer program be taken.

LIST OF REFERENCES

LIST OF REFERENCES

1. Covert, E. E. (editor), et al., "Thrust and Drag: Its Prediction and Verification," Second Printing, New York: American Institute of Aeronautics and Astronautics (AIAA), 1985.

2. Abernethy, R. B. (chairman), et al., "In-Flight Thrust Determination and Uncertainty," Aerospace Technology Conference, Society of Automotive Engineers, SAE SP-674, 1986.

3. Casetti, M., "On the Conventional Definitions of Thrust and Drag of an Aircraft Equipped With a Turbojet Engine," Aircraft Engineering, September 1978.

4. Rosenberg, R. E., and Schuch, G., "The MCA Method of Determining Thrust of Jet Aircraft in Flight," AIAA Journal of Aircraft, Vol 22, No. 10, 1985.

5. Herrington, R. M., et al., "Flight Test Handbook," USAF Technical Report No. 6273, 1966.

6. Nalls, A. L., et al., "Limited A-7D In-Flight Thrust Determination," USAFTPS-LR-85A-SM1, 1985.

7. Perkins, C. D., and Hage, R. E., "Airplane Performance Stability and Control," First Edition, New York: John Wiley & Sons, 1949.

8. Von Mises, R., "Theory of Flight," Paperback Edition, New York: Dover Publications, Inc., 1959.

9. McCormack, B. W., "Aerodynamics, Aeronautics, and Flight Mechanics," First Edition, New York: John Wiley & Sons, Inc., 1979.

10. Cooper, D., "Standard Pascal User Reference Manual," First Edition, New York: W. W. Norton & Company, 1983.

11. Press, W. H., et al., "Numerical Recipes," First Edition, Cambridge: Cambridge University Press, 1986.

12. Rugg, T., and Feldman, P., "Turbo Pascal Program Library," First Edition, Indianapolis: QUE Corp. 1986.

APPENDIXES

APPENDIX 1

Appendix 1 contains Figures 1 through 11 for this study.

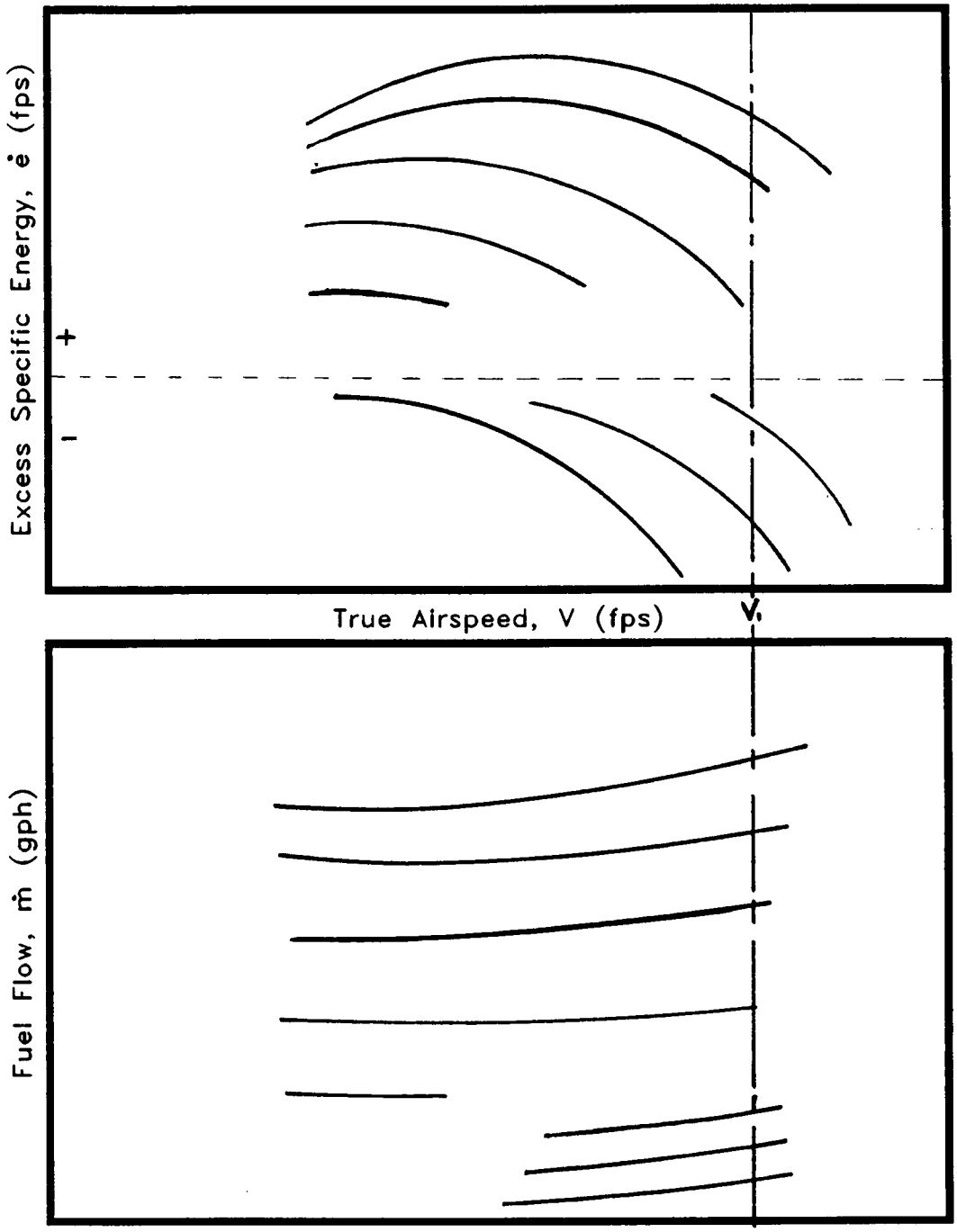

Figure 1. Typical excess specific energy and fuel flow behavior as a function of airspeed for different throttle settings.

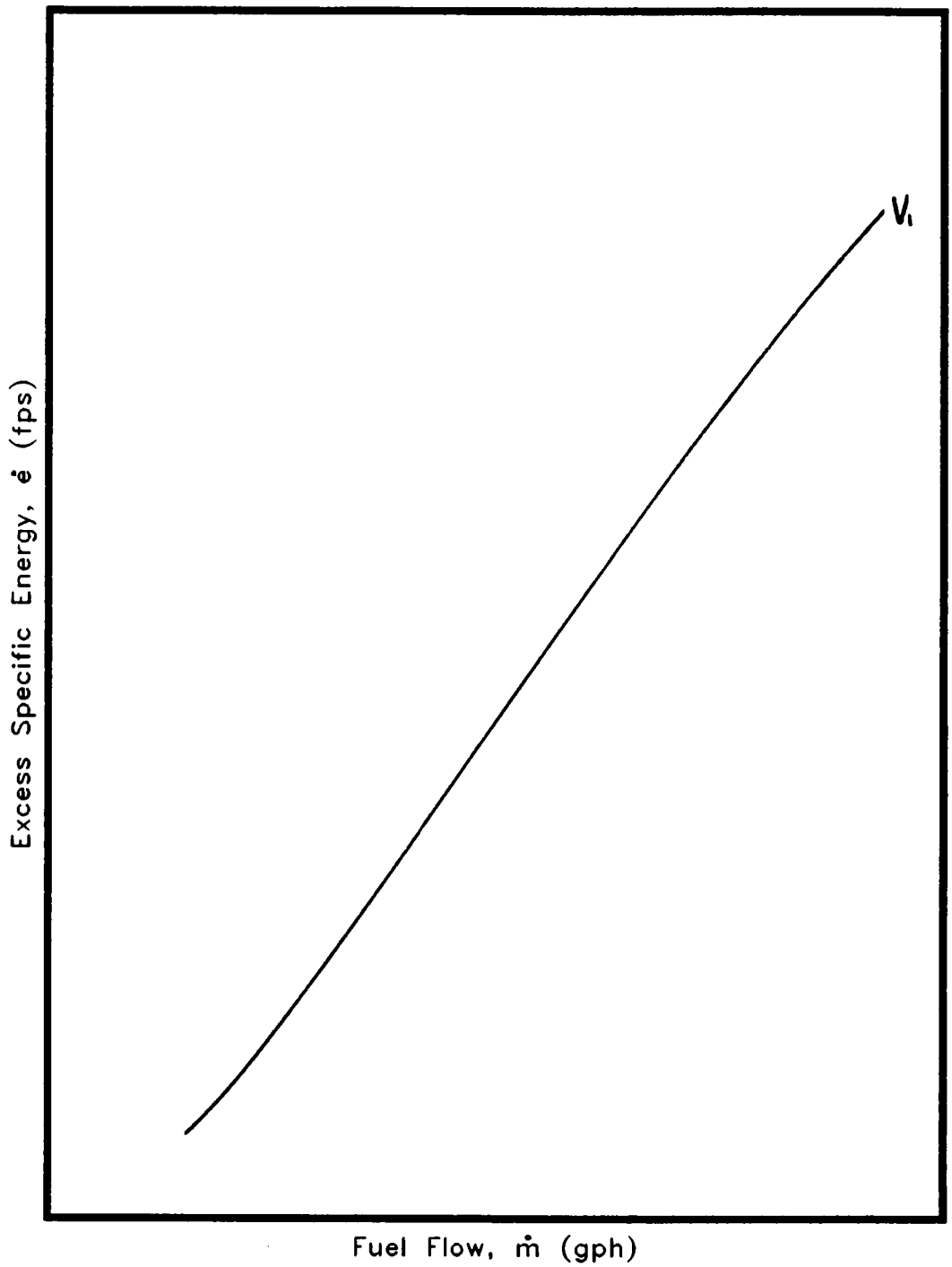

Figure 2. Excess specific energy as a function of fuel flow for a constant airspeed cross-plot.

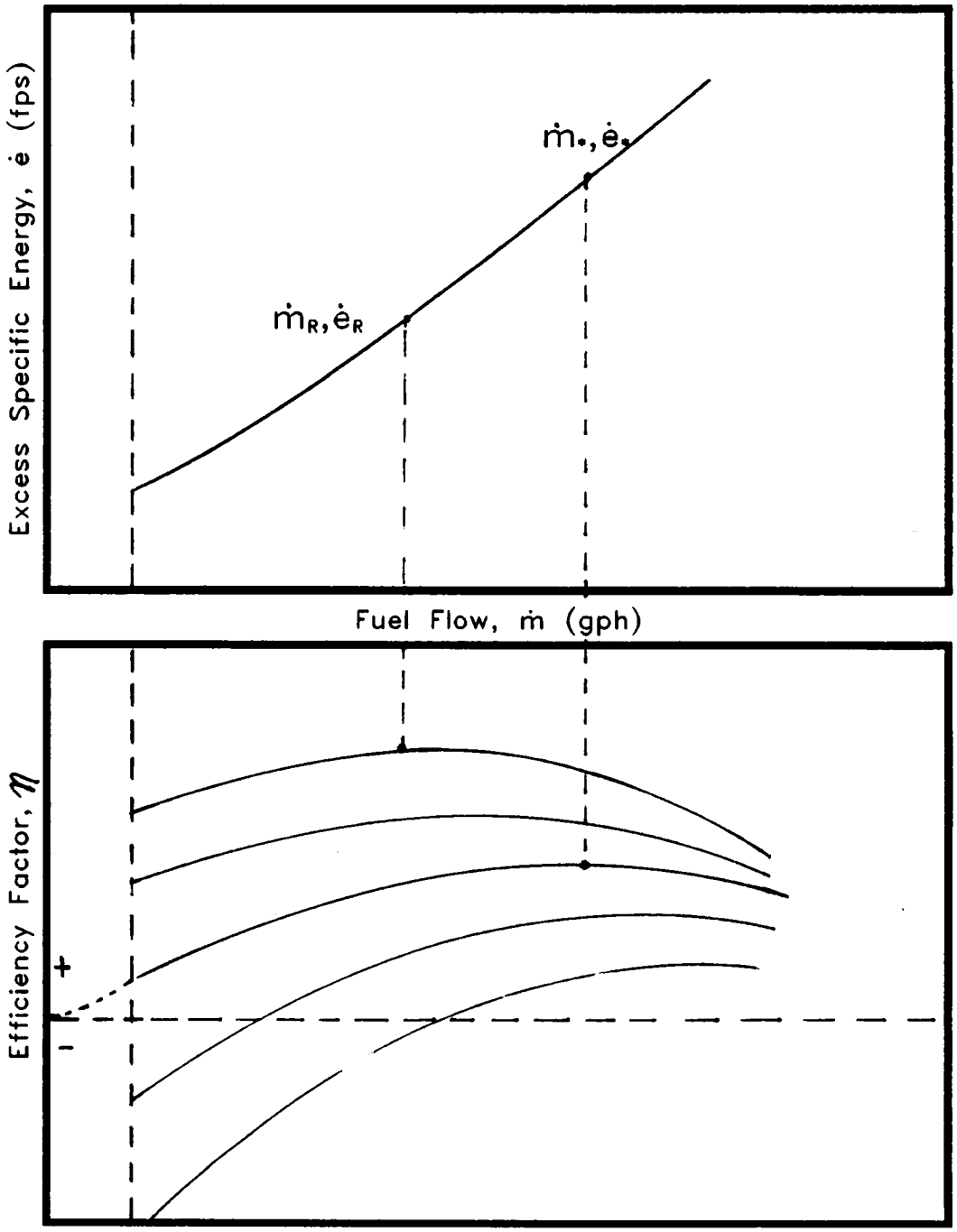

Figure 3. Typical efficiency curves generated as a function of different reference points.

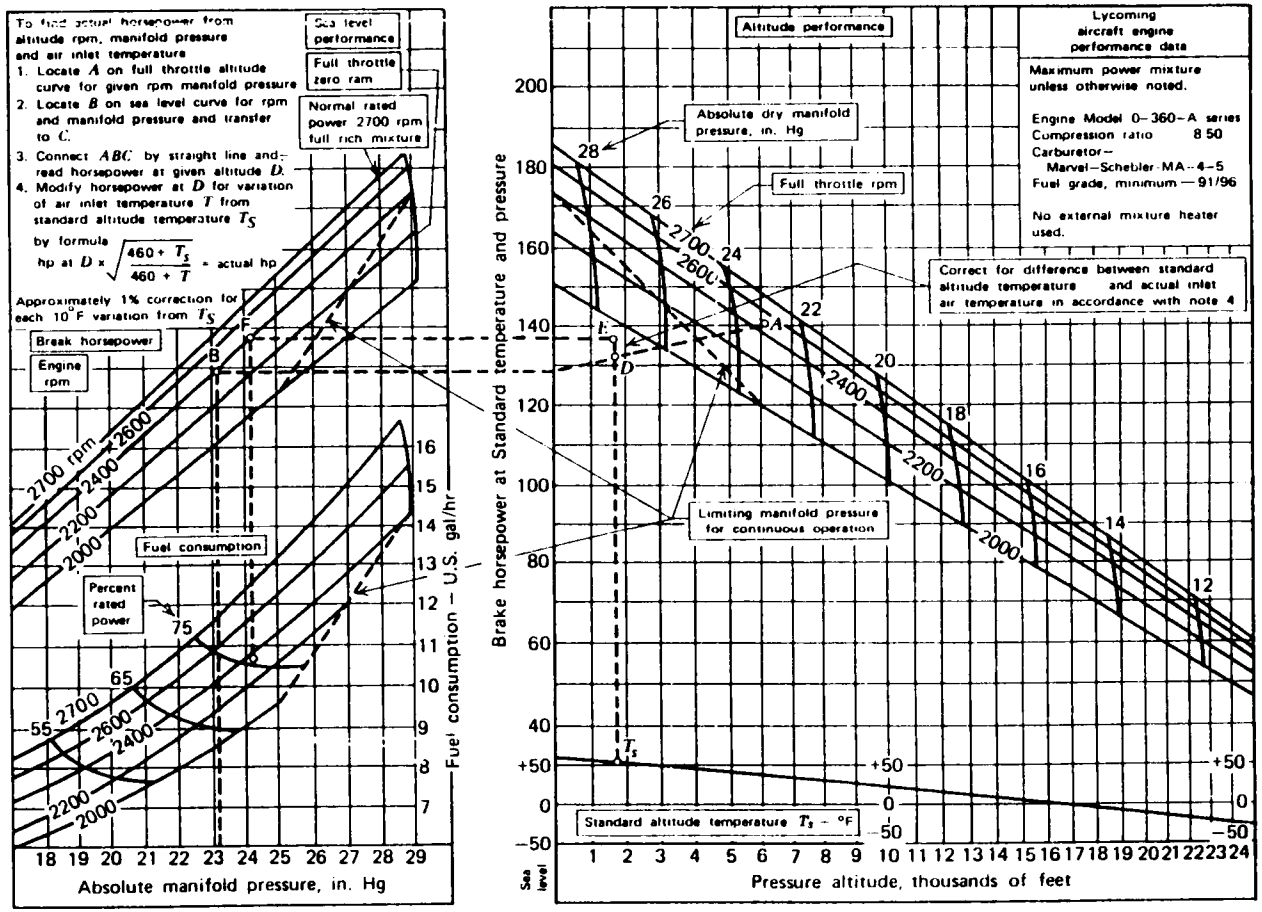

Figure 4. Engine performance chart for O-360-A from (9).

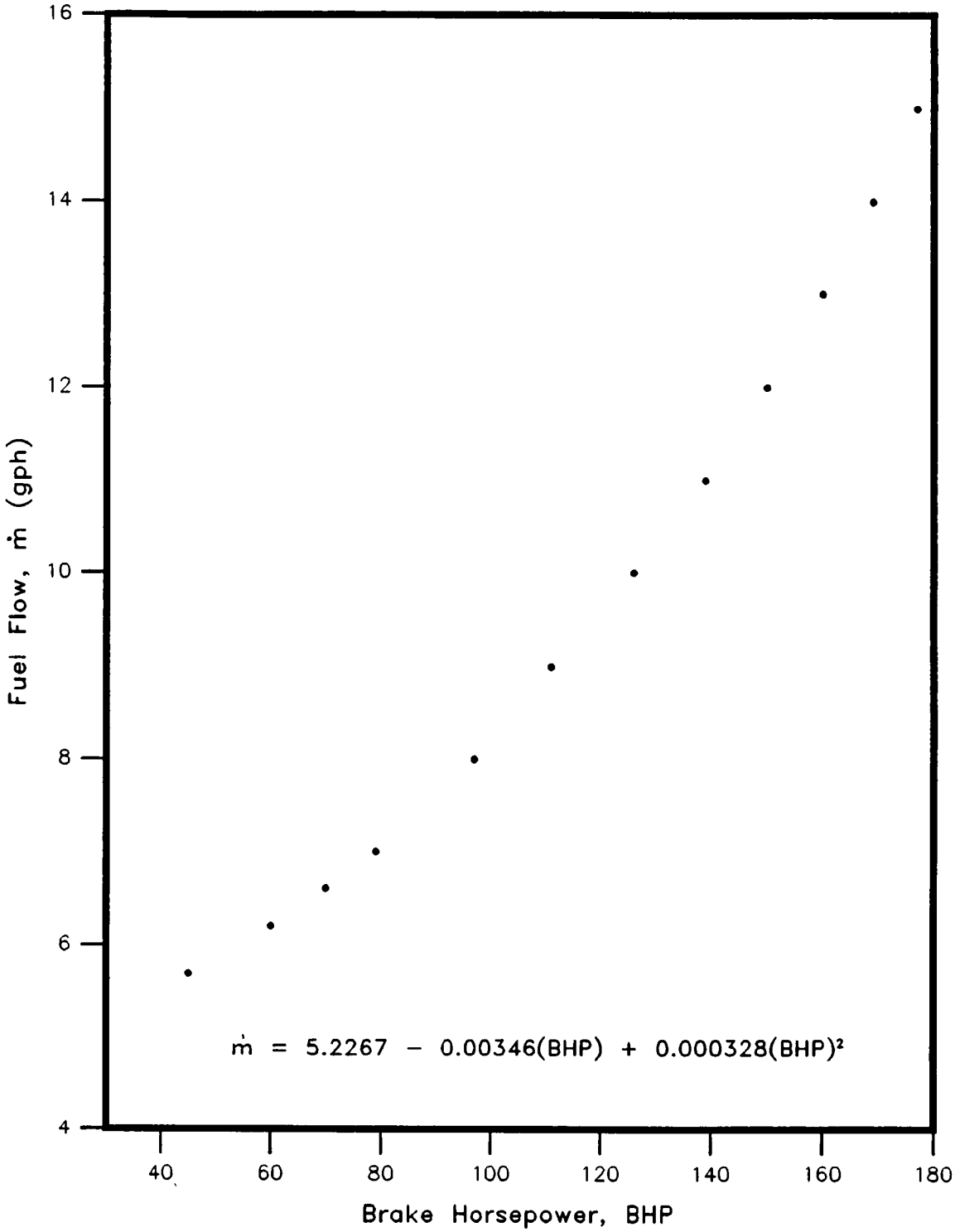

Figure 5. Brake horsepower as a function of fuel flow for the O-360-A engine at sea level standard conditions turning at 2500 rpm.

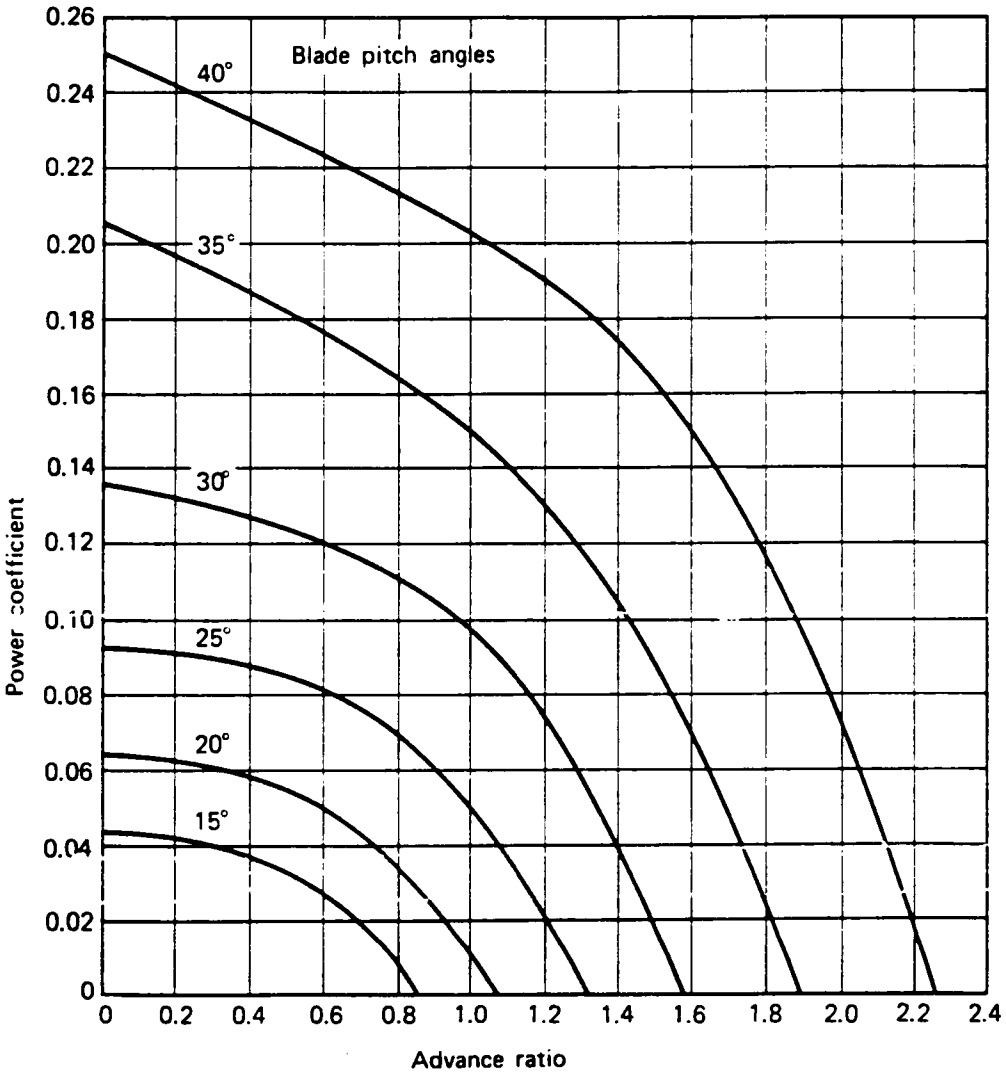

Figure 6. Propeller power coefficients and blade angle model for the Piper Arrow aircraft, reproduced from (9).

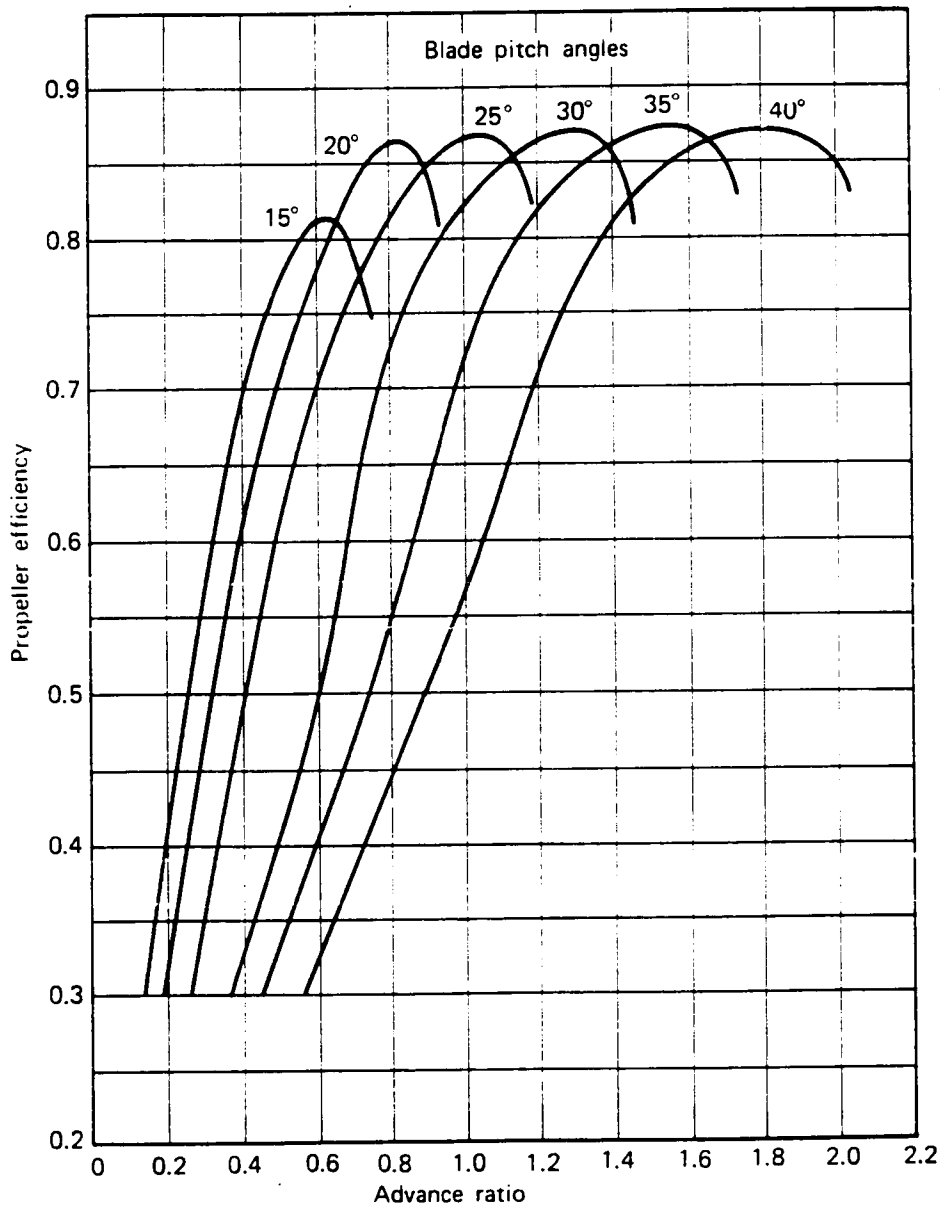

Figure 7. Propeller efficiency model as a function of advance ratio and blade angle for the Piper Arrow Aircraft, reproduced from (9).

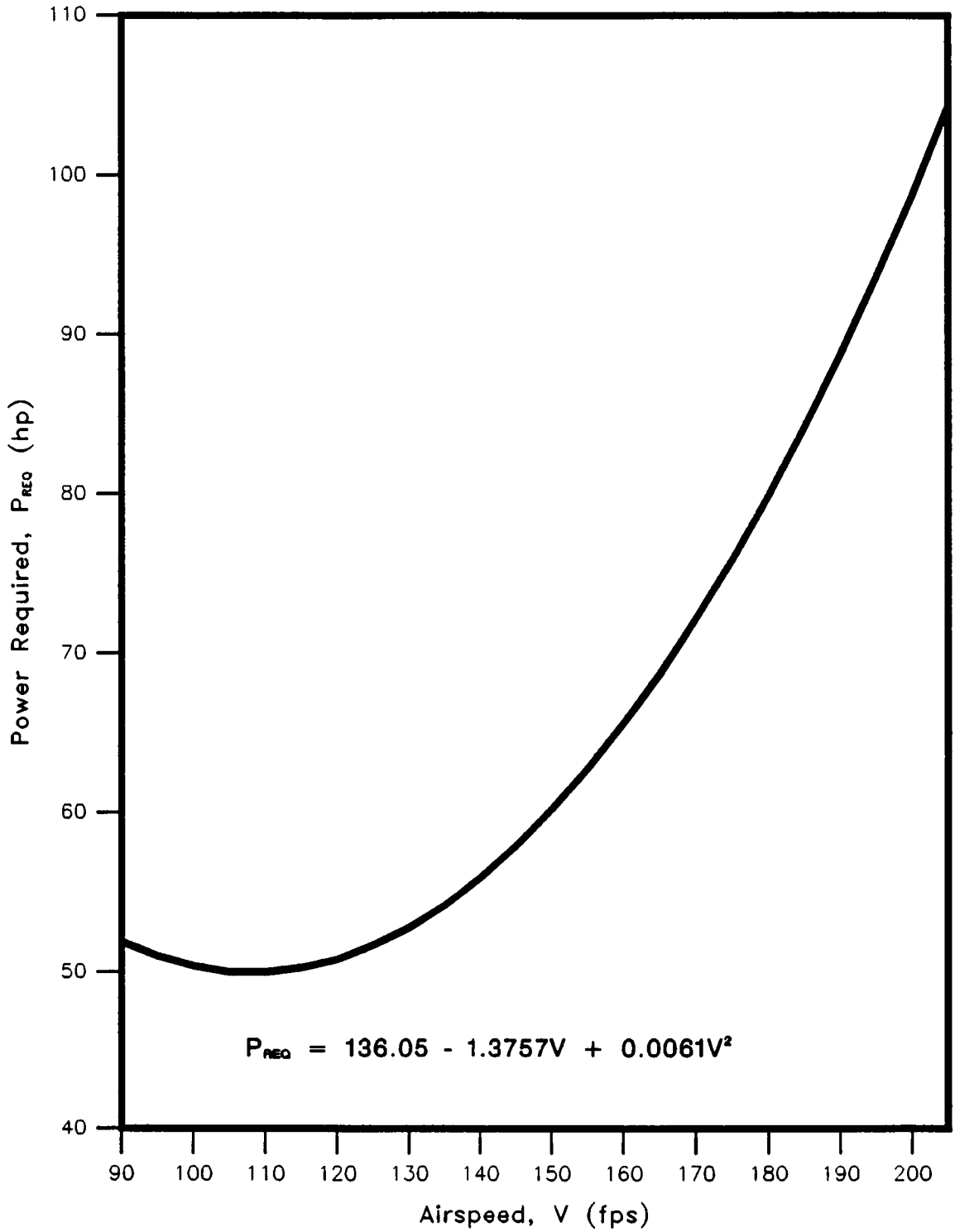

Figure 8. Power required as a function of airspeed for the Piper Arrow at sea level standard conditions and 2500 rpm (9).

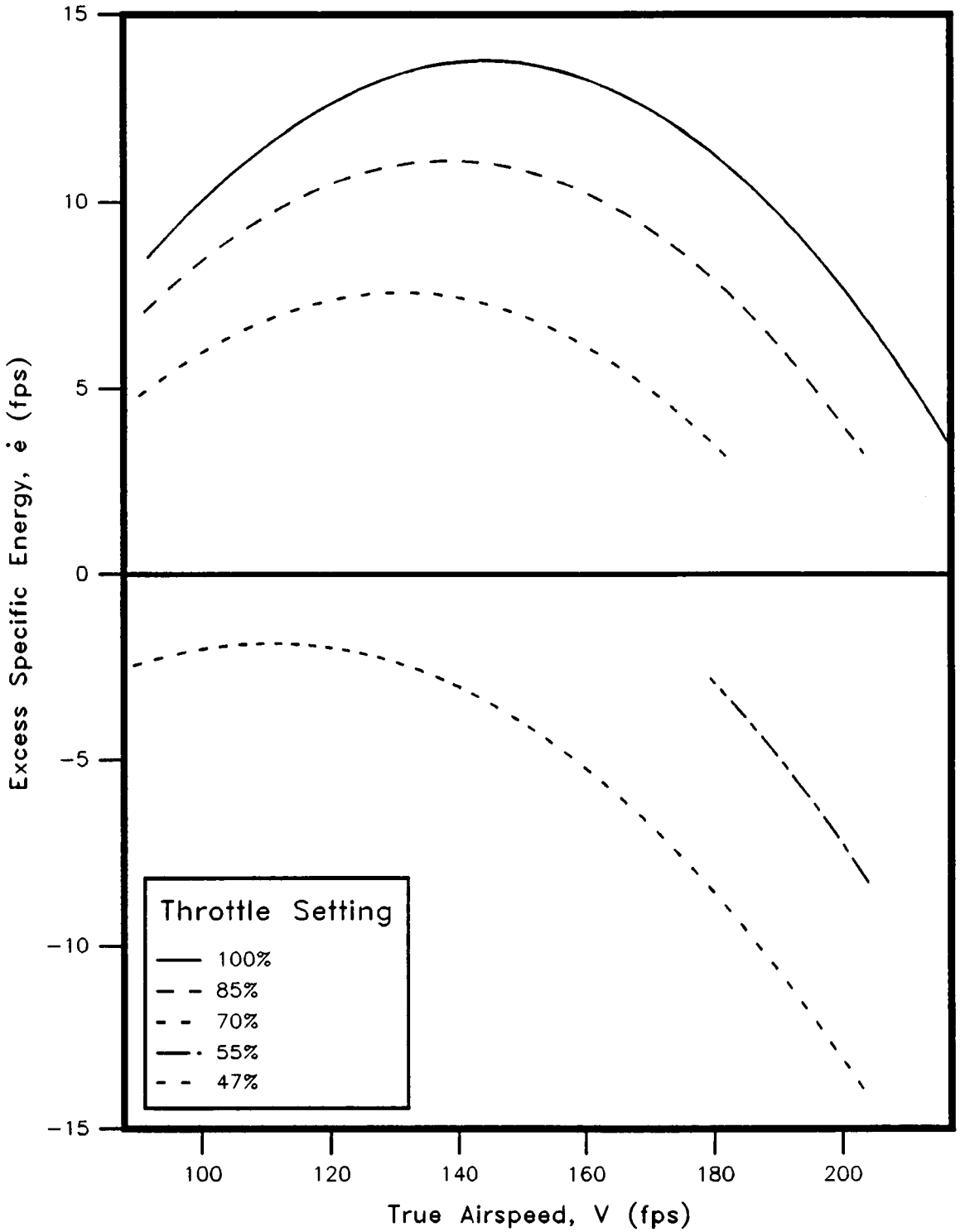

Figure 9. Excess specific energy results as a function of airspeed for selected throttle settings.

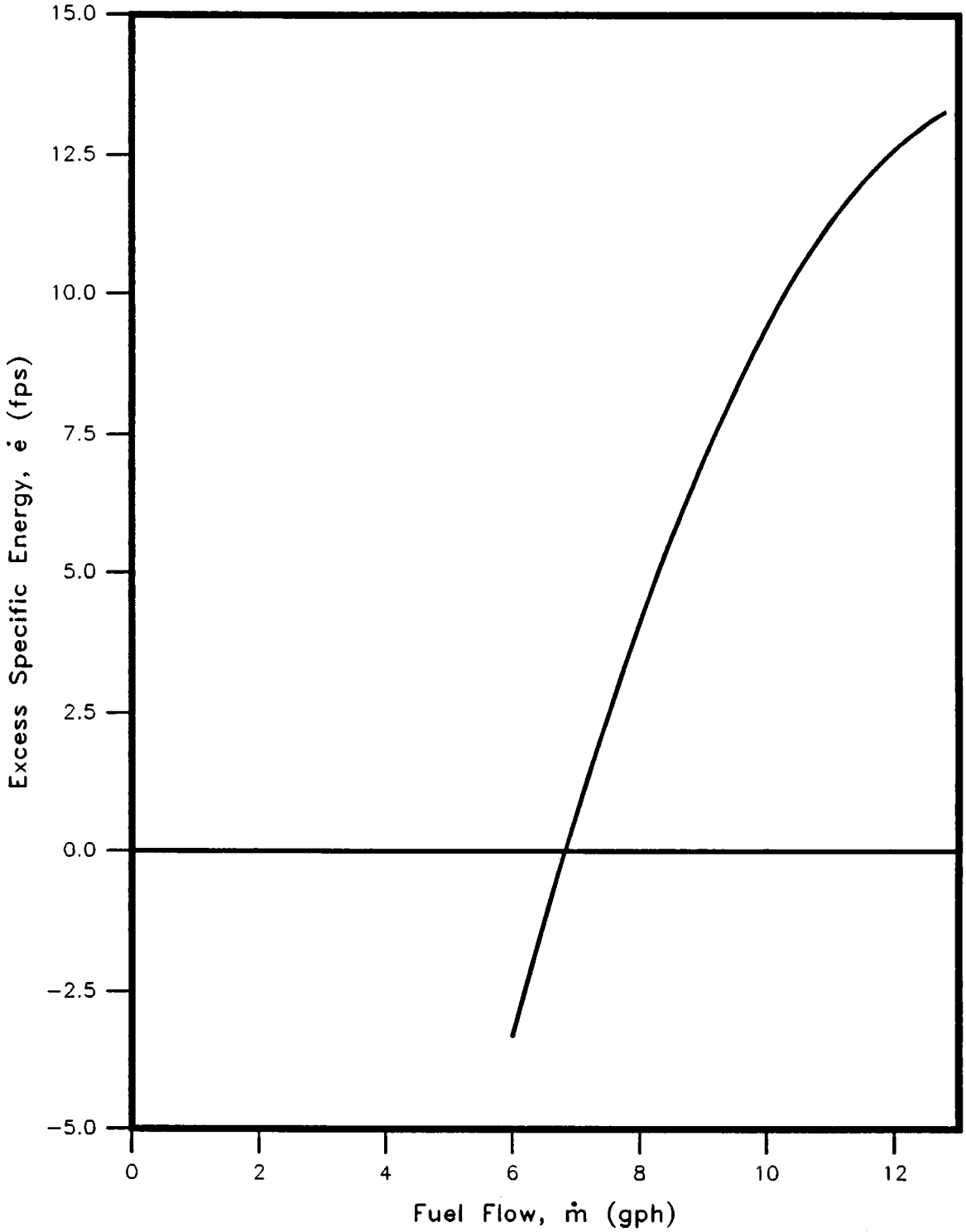

Figure 10. Excess specific energy as a function of fuel flow cross-plotted at an airspeed of 150 fps.

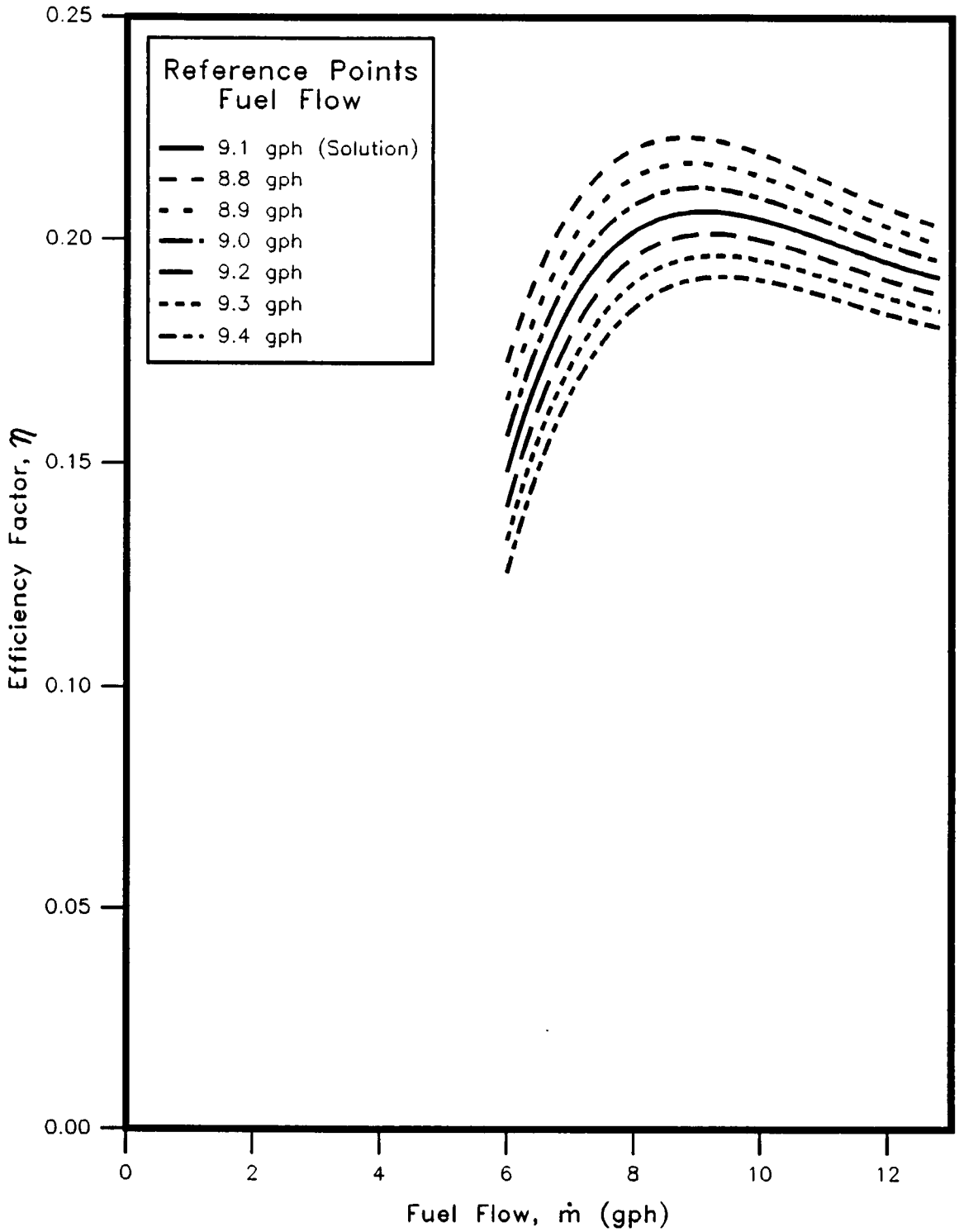

Figure 11. Selected efficiency factor curves generated with reference points from the curve in Figure 10.

APPENDIX 2

PASCAL LISTING OF MCA PROGRAMS

```
program MCA1 (input, output);

{Procedure to generate data for the MCA method.  The}
{procedure models a Piper Arrow with a Lycoming 0-360-A}
{Engine with a standard Weight of 2650 pounds.  Data are}
{produced for level accelerations from stall speed for}
{various throttle settings or for decelerations from an}
{input airspeed.  This version does not use interactive}
{statements so that run time is reduced.  Data are input}
{from a separate command file}

{Coded by Keith Gibby for UTSI Master's Thesis}

{COMPLETED: 20 DEC 1989}
{CHANGE #        DATE              PROCEDURE MODIFIED}

 const                                    {Global declarations}
  StandardWeight  = 2650;                 {Aircraft Weight}
  SeaLevelDensity = 0.0023769;            {slugs/cubic ft}
  SpeedConversion = 3.2810000;            {Meters/Sec-To-Ft/Sec}
  AuxiliaryPower  = 0.000;                {Constant Load}
  GalConversion   = 6.5;                  {Gallons/Hr-->Lbs/Hr}
  ConvHp          = 550;                  {HP-->ft-lb/sec}
  RPS             = 41.6667;              {2500 rpm}
  Diameter        = 6.17;                 {Prop diameter, feet}
  MaxTime         = 150;                  {Maximum allowed Time}
  Degree          = 2;                    {Degree of fit}

 type
  TABLE    = array[0..20] of Real;          {Data Type}
  CURVE    = array [1..MaxTime] of REAL;    {Data Type}
  CpMATRIX = array[0..12, 1..6] of Real;    {Data Type}
  EMATRIX  = array[8..35, 2..10] of Real;   {Data Type}

 var
  SpeedArray          : CURVE;     {Airspeed}
  EffArray            : CURVE;     {Propeller Efficiency}
  CpTAB, Angle        : TABLE;     {Power Coeff, Bladeangle}
  HoldCoeffs          : TABLE;     {Polynomial Coefficients}

  DensityAtAltitude   : TABLE;     {Density Table}
  MaxFuelFlow         : TABLE;     {Max Fuel at Altitude}
  Tolerance           : TABLE;     {Tolerances at Altitude}
  CpArray             : CpMATRIX;  {PowerCoefficient Matrix}
  Etable              : EMATRIX;   {Prop Efficiency Matrix}
```

```
  ActualWeight                  : Real;  {Aircraft Weight}
  DensityRatio, Density         : Real;  {Altitude conditions}
  TrueAirspeed, InitialSpeed    : Real;  {Airspeeds}
  Vew                           : Real;  {Equivalent Airspeed}
  Pew                           : Real;  {Equivalent Power}
  dVt                           : Real;  {Acceleration}
  Edot                          : Real;  {Excess Sp. Energy}
  PowerOut, PowerAvailable      : Real;
  PowerReq, PowerIn, BHP        : Real;
  FuelFlow, InitialFuelFlow     : Real;
  FlowRate, ThrottleSetting     : Real;
  AdvanceRatio, PropE, Cp       : Real;

  MAXNEFF                   : Integer;  {Max Efficiency points}
  AltitudeIndex,Altitude    : Integer;  {Altitude Indentifiers}
  nTests                    : Integer;  {Total Test Runs}
  PropBladeAngle            : Integer;  {Propeller Blade Angle}

  Proceed              : char;          {Switch}
  Accelerate, GLIDE    : BOOLEAN;       {Switch}
  FlightData           : TEXT;          {Output File Type}

 procedure FilePrep;

{Prepare Data File "MCA1.OUT" to be written to}
{COMPLETED: 13 MAR 1989}
{CHANGE #        DATE            MODIFICATION          }


 begin
  OPEN(FlightData, 'MCA1.OUT',history := NEW);
  REWRITE(FlightData);
 end;   {FilePrep}

 procedure InitializeConstants;

{Table for constant values for interpolations and}
{calculations.  Values derived from charts in McCormack}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION          }

 begin
  ActualWeight := 2400;                    {Pounds}

{Altitude Conditions}
  DensityAtAltitude[0] := 0.0023769;
  DensityAtAltitude[2] := 0.0022079;          {2,500 ft}
  DensityAtAltitude[5] := 0.0020482;
  DensityAtAltitude[7] := 0.0018975;          {7,500 ft}
  DensityAtAltitude[10] := 0.0017556;
  DensityAtAltitude[12] := 0.0016219;        {12,500 ft}
```

```
  DensityAtAltitude[13] := 0.0015961;
  DensityAtAltitude[14] := 0.0015455;
  DensityAtAltitude[15] := 0.0014962;
  DensityAtAltitude[20] := 0.0012673;

{Max Fuel Flow Based on Altitude}
  MaxFuelFlow[0] := 12.80;
  MaxFuelFlow[2] := 13.00;              {2,500 ft}
  MaxFuelFlow[5] := 11.80;
  MaxFuelFlow[7] := 10.60;              {7,500 ft}
  MaxFuelFlow[10] := 9.50;
  MaxFuelFlow[12] := 8.80;              {12,500 ft}
  MaxFuelFlow[13] := 8.70;
  MaxFuelFlow[14] := 8.40;
  MaxFuelFlow[15] := 8.00;
  MaxFuelFlow[20] := 6.80;

{Tolerance Table based on Altitude}
  Tolerance[0]  := 0.50;
  Tolerance[2]  := 1.00;
  Tolerance[5]  := 1.00;
  Tolerance[7]  := 1.00;
  Tolerance[10] := 0.50;
  Tolerance[12] := 0.20;
  Tolerance[13] := 0.20;
  Tolerance[14] := 0.10;
  Tolerance[15] := 0.10;
  Tolerance[20] := 0.10;

{Power Coefficient vs Advance Ratio for Blade Angle}
{Of the form CpArray[AdvanceRatio*10,(BladeAngle-10)/5}
  CpArray[2, 1] := 0.042;
  CpArray[2, 2] := 0.064;
  CpArray[2, 3] := 0.091;
  CpArray[2, 4] := 0.132;
  CpArray[2, 5] := 0.197;
  CpArray[2, 6] := 0.242;
  CpArray[4, 1] := 0.037;
  CpArray[4, 2] := 0.058;
  CpArray[4, 3] := 0.088;
  CpArray[4, 4] := 0.127;
  CpArray[4, 5] := 0.187;
  CpArray[4, 6] := 0.233;
  CpArray[6, 1] := 0.027;
  CpArray[6, 2] := 0.050;
  CpArray[6, 3] := 0.081;
  CpArray[6, 4] := 0.120;
  CpArray[6, 5] := 0.176;
  CpArray[6, 6] := 0.223;
  CpArray[8, 1] := 0.008;
  CpArray[8, 2] := 0.034;
  CpArray[8, 3] := 0.070;
```

```
  CpArray[8, 4] := 0.111;
  CpArray[8, 5] := 0.164;
  CpArray[8, 6] := 0.214;
  CpArray[10, 1] := -0.020;
  CpArray[10, 2] := 0.011;
  CpArray[10, 3] := 0.051;
  CpArray[10, 4] := 0.098;
  CpArray[10, 5] := 0.150;
  CpArray[10, 6] := 0.203;

{Prop Efficiency vs Blade Angle for constant Advance Ratio}
{Of the form Etable[BladeAngle,AdvanceRatio*10].}
{-1 denotes no value}
  Etable[9, 2] := 0.51;
  Etable[9, 3] := 0.64;
  Etable[9, 4] := 0.74;
  Etable[9, 5] := 0.75;
  Etable[9, 6] := -1;
  Etable[9, 7] := -1;
  Etable[9, 8] := -1;
  Etable[9, 9] := -1;
  Etable[9, 10] := -1;
  Etable[10, 2] := 0.50;
  Etable[10, 3] := 0.63;
  Etable[10, 4] := 0.74;
  Etable[10, 5] := 0.75;
  Etable[10, 6] := 0.70;
  Etable[10, 7] := -1;
  Etable[10, 8] := -1;
  Etable[10, 9] := -1;
  Etable[10, 10] := -1;
  Etable[11, 2] := 0.48;
  Etable[11, 3] := 0.63;
  Etable[11, 4] := 0.74;
  Etable[11, 5] := 0.76;
  Etable[11, 6] := 0.72;
  Etable[11, 7] := -1;
  Etable[11, 8] := -1;
  Etable[11, 9] := -1;
  Etable[11, 10] := -1;
  Etable[12, 2] := 0.47;
  Etable[12, 3] := 0.61;
  Etable[12, 4] := 0.73;
  Etable[12, 5] := 0.77;
  Etable[12, 6] := 0.74;
  Etable[12, 7] := 0.70;
  Etable[12, 8] := -1;
  Etable[12, 9] := -1;
  Etable[12, 10] := -1;
  Etable[13, 2] := 0.45;
  Etable[13, 3] := 0.59;
  Etable[13, 4] := 0.72;
```

```
Etable[13, 5] := 0.78;
Etable[13, 6] := 0.76;
Etable[13, 7] := 0.73;
Etable[13, 8] := -1;
Etable[13, 9] := -1;
Etable[13, 10] := -1;
Etable[14, 2] := 0.44;
Etable[14, 3] := 0.58;
Etable[14, 4] := 0.71;
Etable[14, 5] := 0.77;
Etable[14, 6] := 0.79;
Etable[14, 7] := 0.75;
Etable[14, 8] := -1;
Etable[14, 9] := -1;
Etable[14, 10] := -1;
Etable[15, 2] := 0.42;
Etable[15, 3] := 0.57;
Etable[15, 4] := 0.70;
Etable[15, 5] := 0.77;
Etable[15, 6] := 0.81;
Etable[15, 7] := 0.77;
Etable[15, 8] := 0.71;
Etable[15, 9] := -1;
Etable[15, 10] := -1;
Etable[16, 2] := 0.40;
Etable[16, 3] := 0.55;
Etable[16, 4] := 0.68;
Etable[16, 5] := 0.76;
Etable[16, 6] := 0.81;
Etable[16, 7] := 0.80;
Etable[16, 8] := 0.75;
Etable[16, 9] := -1;
Etable[16, 10] := -1;
Etable[17, 2] := 0.39;
Etable[17, 3] := 0.53;
Etable[17, 4] := 0.67;
Etable[17, 5] := 0.74;
Etable[17, 6] := 0.80;
Etable[17, 7] := 0.82;
Etable[17, 8] := 0.79;
Etable[17, 9] := 0.74;
Etable[17, 10] := -1;
Etable[18, 2] := 0.37;
Etable[18, 3] := 0.51;
Etable[18, 4] := 0.65;
Etable[18, 5] := 0.73;
Etable[18, 6] := 0.80;
Etable[18, 7] := 0.84;
Etable[18, 8] := 0.82;
Etable[18, 9] := 0.77;
Etable[18, 10] := -1;
Etable[19, 2] := 0.35;
```

```
Etable[19, 3] := 0.49;
Etable[19, 4] := 0.63;
Etable[19, 5] := 0.72;
Etable[19, 6] := 0.79;
Etable[19, 7] := 0.84;
Etable[19, 8] := 0.85;
Etable[19, 9] := 0.80;
Etable[19, 10] := -1;
Etable[20, 2] := 0.33;
Etable[20, 3] := 0.47;
Etable[20, 4] := 0.61;
Etable[20, 5] := 0.70;
Etable[20, 6] := 0.78;
Etable[20, 7] := 0.83;
Etable[20, 8] := 0.86;
Etable[20, 9] := 0.84;
Etable[20, 10] := -1;
Etable[21, 2] := 0.31;
Etable[21, 3] := 0.45;
Etable[21, 4] := 0.59;
Etable[21, 5] := 0.69;
Etable[21, 6] := 0.77;
Etable[21, 7] := 0.82;
Etable[21, 8] := 0.86;
Etable[21, 9] := 0.86;
Etable[21, 10] := -1;
Etable[22, 2] := 0.29;
Etable[22, 3] := 0.43;
Etable[22, 4] := 0.57;
Etable[22, 5] := 0.67;
Etable[22, 6] := 0.75;
Etable[22, 7] := 0.81;
Etable[22, 8] := 0.85;
Etable[22, 9] := 0.87;
Etable[22, 10] := 0.82;
Etable[23, 2] := 0.27;
Etable[23, 3] := 0.40;
Etable[23, 4] := 0.55;
Etable[23, 5] := 0.65;
Etable[23, 6] := 0.74;
Etable[23, 7] := 0.80;
Etable[23, 8] := 0.84;
Etable[23, 9] := 0.86;
Etable[23, 10] := 0.85;
Etable[24, 3] := 0.38;
Etable[24, 4] := 0.53;
Etable[24, 5] := 0.63;
Etable[24, 6] := 0.72;
Etable[24, 7] := 0.78;
Etable[24, 8] := 0.83;
Etable[24, 9] := 0.85;
Etable[24, 10] := 0.87;
```

```
Etable[25, 2] := 0.23;
Etable[25, 3] := 0.35;
Etable[25, 4] := 0.50;
Etable[25, 5] := 0.61;
Etable[25, 6] := 0.70;
Etable[25, 7] := 0.76;
Etable[25, 8] := 0.82;
Etable[25, 9] := 0.84;
Etable[25, 10] := 0.87;
Etable[26, 2] := 0.21;
Etable[26, 3] := 0.33;
Etable[26, 4] := 0.46;
Etable[26, 5] := 0.59;
Etable[26, 6] := 0.67;
Etable[26, 7] := 0.74;
Etable[26, 8] := 0.80;
Etable[26, 9] := 0.83;
Etable[26, 10] := 0.86;
Etable[27, 2] := -1;
Etable[27, 3] := 0.31;
Etable[27, 4] := 0.43;
Etable[27, 5] := 0.54;
Etable[27, 6] := 0.64;
Etable[27, 7] := 0.72;
Etable[27, 8] := 0.78;
Etable[27, 9] := 0.82;
Etable[27, 10] := 0.85;
Etable[28, 2] := -1;
Etable[28, 3] := 0.29;
Etable[28, 4] := 0.39;
Etable[28, 5] := 0.49;
Etable[28, 6] := 0.60;
Etable[28, 7] := 0.69;
Etable[28, 8] := 0.76;
Etable[28, 9] := 0.81;
Etable[28, 10] := 0.84;
Etable[29, 2] := -1;
Etable[29, 3] := 0.27;
Etable[29, 4] := 0.36;
Etable[29, 5] := 0.44;
Etable[29, 6] := 0.55;
Etable[29, 7] := 0.66;
Etable[29, 8] := 0.75;
Etable[29, 9] := 0.80;
Etable[29, 10] := 0.83;
Etable[30, 2] := -1;
Etable[30, 3] := 0.25;
Etable[30, 4] := 0.33;
Etable[30, 5] := 0.41;
Etable[30, 6] := 0.50;
Etable[30, 7] := 0.63;
Etable[30, 8] := 0.73;
```

```
 Etable[30, 9] := 0.78;
 Etable[30, 10] := 0.82;
 Etable[31, 2] := -1;
 Etable[31, 3] := 0.24;
 Etable[31, 4] := 0.31;
 Etable[31, 5] := 0.39;
 Etable[31, 6] := 0.47;
 Etable[31, 7] := 0.59;
 Etable[31, 8] := 0.70;
 Etable[31, 9] := 0.76;
 Etable[31, 10] := 0.81;
 Etable[32, 2] := -1;
 Etable[32, 3] := 0.23;
 Etable[32, 4] := 0.29;
 Etable[32, 5] := 0.37;
 Etable[32, 6] := 0.45;
 Etable[32, 7] := 0.53;
 Etable[32, 8] := 0.67;
 Etable[32, 9] := 0.73;
 Etable[32, 10] := 0.79;
 Etable[33, 2] := -1;
 Etable[33, 3] := 0.22;
 Etable[33, 4] := 0.28;
 Etable[33, 5] := 0.36;
 Etable[33, 6] := 0.43;
 Etable[33, 7] := 0.49;
 Etable[33, 8] := 0.63;
 Etable[33, 9] := 0.70;
 Etable[33, 10] := 0.77;
 Etable[34, 2] := -1;
 Etable[34, 3] := 0.21;
 Etable[34, 4] := 0.27;
 Etable[34, 5] := 0.34;
 Etable[34, 6] := 0.41;
 Etable[34, 7] := 0.45;
 Etable[34, 8] := 0.59;
 Etable[34, 9] := 0.67;
 Etable[34, 10] := 0.75;
 Etable[35, 2] := -1;
 Etable[35, 3] := 0.20;
 Etable[35, 4] := 0.26;
 Etable[35, 5] := 0.33;
 Etable[35, 6] := 0.39;
 Etable[35, 7] := 0.43;
 Etable[35, 8] := 0.55;
 Etable[35, 9] := 0.63;
 Etable[35, 10] := 0.72;
end; {InitializeConstants}

function XtoY (Base, Exponent : Real) : Real;
```

```
{Math tool to raise base X to power Y}
{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

 begin
  XtoY := exp(Exponent * Ln(Base));
 end;  {XtoY}


 function Interpolate (xa, ya : TABLE;
         k : integer;
         x : Real) : Real;

{Interpolation Routine.  xa,ya are input array, k is number}
{of data points}
{x is the point of interest}

{Ref: "NUMERICAL RECIPES", Press, Flannery, Teulolsky, and}
{Vetterling}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

  var
   c, d : TABLE;
   ns, m, i : integer;
   w, hp, ho, dift, dif, den, dy, y : Real;
 begin
  ns := 1;
  dif := ABS(x - xa[1]);

  for i := 1 to k do
   begin
    dift := ABS(x - xa[i]);

    if (dift < dif) then
     begin
      ns := i;
      dif := dift
     end;  {IF}

    c[i] := ya[i];
    d[i] := ya[i];
   end;  {FOR}

  y := ya[ns];
  ns := ns - 1;

  for m := 1 to k - 1 do
   begin

    for i := 1 to k - m do
```

```
      begin
       ho := xa[i] - x;
       hp := xa[i + m] - x;
       w := c[i + 1] - d[i];
       den := ho - hp;

       if (den = 0.0) then
        begin
         writeln('Pause in Interpolator Routine');
         readln
        end; {IF}

       den := w / den;
       d[i] := hp * den;
       c[i] := ho * den
      end;  {FOR}

     if ((2 * ns) < (k - m)) then
      begin
       dy := c[ns + 1]
      end  {IF}

     else
      begin
       dy := d[ns];
       ns := ns - 1
      end; {END}

     y := y + dy
    end;  {BIG FOR}

   Interpolate := y;

 end;  {function Interpolate}


Procedure PolySmooth(XArray,YArray:CURVE;
                     NumDataPairs:integer);

{Polynomial Curve Fit Routine.}

{Adapted from "TURBO PASCAL PROGRAM LIBRARY", by Rugg and}
{Feldman}

const
   RowSize   = 250;
   ColSize   = 2;
   MaxDegree = 7;

type
   CoeffArrayType = array[0..30] of real;
```

```
var
   PowerArray, RHS, Coeffs : CoeffArrayType;
   Factors                 : array[1..15, 1..15] of real;
   TwoDegree, NumEqns, Code : integer;
   J, K, L, M, Last                                  : integer;
   CharFlag                                          : char;
   TimeToQuit, NoMoreToDo                            : boolean;
   Work, Numer, Denom, Correlation, MeanY, Y    : real;

procedure pInterpolate;

begin
  Y := 0.0;
  for K := 1 to NumEqns do
   begin
    HoldCoeffs[K] := Coeffs[K]; {Get Coefficients}
   end;  {for}
end;

procedure SolveEqns;

begin
   TwoDegree := Degree * 2;
   NumEqns   := Degree + 1;
   for J := 0 to TwoDegree do
      begin
         PowerArray[J] := 0.0;
         for K := 1 to NumDataPairs do
            PowerArray[J] := PowerArray[J]+XtoY(XArray[K],J)
      end;
   for J := 1 to NumEqns do
      begin
         RHS[J] := 0.0;
         for K := 1 to NumDataPairs do
            RHS[J] := RHS[J] + YArray[K] *
                              XtoY(XArray[K], J - 1)
      end;
   for J := 1 to NumEqns do
      for K := 1 to NumEqns do
         Factors[J,K] := PowerArray[J + K - 2];
   for K := 1 to Degree do
      begin
         M := K + 1;
         L := K;
         repeat
            if abs(Factors[M,K]) > abs(Factors[L,K]) then
               L := M;
            M := M + 1
         until
            M > NumEqns;
         for J := K to NumEqns do
            begin
```

```
                Work          := Factors[K,J];
                Factors[K,J] := Factors[L,J];
                Factors[L,J] := Work;
             end;
          Work := RHS[K];  RHS[K] := RHS[L];  RHS[L] := Work;
          M := K + 1;
          repeat
             Work          := Factors[M,K] / Factors[K,K];
             Factors[M,K] := 0.0;
             for J := K + 1 to NumEqns do
                Factors[M,J] := Factors[M,J] -
                                Work * Factors[K,J];
             RHS[M] := RHS[M] - Work * RHS[K];
             M      := M + 1
          until
             M > NumEqns
       end;
    Coeffs[NumEqns] := RHS[NumEqns] /
                       Factors[NumEqns,NumEqns];
    for M := Degree downto 1 do
       begin
          Work := 0.0;
          for J := M + 1 to NumEqns do
             begin
                Work      := Work + Factors[M,J] * Coeffs[J];
                Coeffs[M] := (RHS[M] - Work) / Factors[M,M]
             end
       end;
    Numer := 0.0;
    Denom := 0.0;
    for J := 1 to NumDataPairs do
       begin
          Work := 0.0;
          for K := 1 to NumEqns do
             Work := Work+Coeffs[K] * XtoY(XArray[J], K - 1);
          Numer := Numer + sqr(YArray[J] - Work);
          Denom := Denom + sqr(YArray[J] - MeanY)
       end;
    if Denom = 0.0 then
       Correlation := 1.0
    else
       Correlation := sqrt(1.0 - Numer / Denom);
end;

BEGIN
   SolveEqns;
   pInterpolate;
END;   {PolySmooth}


 function StallSpeed : Real;
```

```
{Calculate Stall speed,in ft/sec based on linear fit}
{Reference McCormick: p. 461}
{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

  const
   m = 1.558475e-3;    {Slope}
   b = 88.6;           {Intercept}

 begin
  StallSpeed := m * Altitude + b;
 end;   {StallSpeed}


 function FuelPower (Option : Integer) : Real;

{Calculate Power or Fuel Flow}
{ 0 to compute BHP }
{ 1 to compute Fuel Required}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

  var
   C1, C2 : Real;

 begin
  if Option = 1 then {Curve Fit for Power vs Fuel Flow}
   FuelPower := 5.2267-0.0034643*BHP+0.00032788 * SQR(BHP);

  if Option = 0 then {Solve Quadratic Equation for BHP}
   begin
    C1 := SQRT(SQR(0.0034643) - 4.000000 * 0.00032788 *
          (5.2267 - FlowRate));
    C2 := -0.0034643 - C1;
    FuelPower := ABS(C2 / (2 * 0.00032788));
   end;  {IF}

 end;  {FuelPower}


 function THP : Real;

{2nd Order Least Squares Curve Fit of Power Required Chart}
{(Fig 7.15,McCormack)}
{Uses Vew as Independent Variable.  Vew Must be in Ft/Sec}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}


  const
```

```
   A0 = 136.0536855;
   A1 = -1.375733333;
   A2 = 6.06197163e-3;

 begin
  THP := A0 + A1 * Vew + A2 * SQR(Vew);
 end;  {THP}


 procedure SetAltitude;

{Define Test Altitude}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

 begin
  {WriteLn('Enter Altitude in Thousands of Feet
          (0,5,15,20)K:');}
  ReadLn(AltitudeIndex);
  Altitude := AltitudeIndex * 1000;
  DensityRatio := DensityAtAltitude[AltitudeIndex] /
                  SeaLevelDensity;
  Density := DensityAtAltitude[AltitudeIndex];
 end;    {SetAltitude}


 function WeightRatio : Real;

{Calculate Weight ratio}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

 begin
  WeightRatio := StandardWeight / ActualWeight;
 end;   {WeightRatio}


 procedure Airspeed;

{Define Airspeed, either stall speed or maximum airspeed}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

  var
   PASS : integer;     {0 or 1}

 begin
  {WriteLn('To set airspeed enter 0, or enter 1 for stall
           speed');}
```

```
 ReadLn(PASS);

 if PASS = 0 then {Initial speed point for deceleration}
  begin
   {WriteLn('Enter True Airspeed (ft/sec):');}
   ReadLn(TrueAirspeed);
   InitialSpeed := TrueAirspeed;
  end  {IF}

 else
  begin
   TrueAirspeed := StallSpeed;
   InitialSpeed := TrueAirspeed;
  end; {else}

end;  {Airspeed}


procedure PowerRequired;

{Determine actual power required from prop by correcting}
{Standard Day Power Required vs Velocity Charts for weight}
{and altitude}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

begin
 Vew := TrueAirspeed * SQRT(DensityRatio) *
        SQRT(WeightRatio);
 Pew := THP;  {THP uses Vew to get THP Required}
 PowerReq := Pew/(SQRT(DensityRatio)
             * XtoY(WeightRatio,1.5));
end;  {PowerRequired}


function BladeAngle : Real;

{Determine Prop Blade Angle from Advance Ratio and Power}
{Coefficient}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

 var
  Slope : Real;
  n : integer;     {Counter}
  j : integer;     {Integer value of Advance Ratio}

begin
 if ODD(TRUNC(10 * AdvanceRatio)) then
  j := (TRUNC(10 * AdvanceRatio)) - 1
```

```
  else
   j := (TRUNC(10 * AdvanceRatio));

  for n := 1 to 6 do
   begin
    slope := (CpArray[j + 2, n] - CpArray[j, n]) / ((j + 2)
             / 10 - (j / 10));
    CpTAB[n] := CpArray[j + 2, n] - Slope * (((j + 2) / 10)
                - AdvanceRatio);
    Angle[n] := 10 + n * 5;
   end;  {FOR}

  BladeAngle := Interpolate(CpTAB, Angle, 6, Cp);

 end; {function BladeAngle}


 function PropellerEfficiency : Real;

{Compute prop efficiency from Advance Ratio and Blade Angle}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE              MODIFICATION}

  var
   Xj, Ye             : TABLE;   {Data Array}
   variables, Jindex  : integer; {Counters}
   Emanual            : Real;    {User defined Efficiency}

 begin
  PropBladeAngle := ROUND(BladeAngle);
  variables := 0;

  for Jindex := 2 to 10 do
   begin

    if Etable[PropBladeAngle, Jindex] > 0 then
     begin
      variables := variables + 1;
      Xj[variables] := Jindex / 10;
      Ye[variables] := Etable[PropBladeAngle, Jindex];
     end; {IF}

   end; {FOR}

  if AdvanceRatio > Xj[variables] then        {Warn User}
   begin
    WriteLn('AdvanceRatio Out of Range:',AdvanceRatio:1:3);
    WriteLn('Propeller Blade Angle= ', PropBladeAngle);
    PropellerEfficiency := 2;          {Out of Range}
    WriteLn;
   end {IF}
```

```
  else
    PropellerEfficiency := Interpolate(Xj, Ye, Variables,
                                       AdvanceRatio);

 end; {PropellerEfficiency}


 procedure SmoothedPropellerEfficiency(mode:integer);

{Interpolated values are smoothed if mode = 0. If mode = 1}
{then values are generated from smoothing coefficients}

 {COMPLETED: 22 FEB 1990}
 {CHANGE #        DATE              MODIFICATION}

  var
   TestEff                : real;    {Efficiency Value}
   SpeedIndex             : integer; {Airspeed Index}
   Neff                   : integer; {Efficiency Points}
   Inbounds               : BOOLEAN; {Airspeed Boundary}

  function PropEquations : real;
  {Use coefficients from curve fit to generate efficiencies}

  {COMPLETED: 22 FEB 1990}
  {CHANGE #        DATE              MODIFICATION}

   var
    z : integer;   {Counter}
    y : real;      {Solution}

   begin
    y := 0;

    for z := 1 to (Degree + 1) do
     y := y + HoldCoeffs[z]* XtoY(TrueAirspeed,z-1);

    PropEquations := y;
   end;   {PropEquations}


  begin
    if mode=0 then      {Generate efficiency data table}
     begin
      Neff:=0;          {Number of efficiency points}
      Inbounds := TRUE;
      SpeedIndex := 84;    {Airspeed lower limit}

      while Inbounds do
       begin
         Neff:= Neff+1;
         MaxNeff := Neff;
```

```
        SpeedIndex := SpeedIndex + 1;

        if SpeedIndex >= 220 then Inbounds := FALSE;

        SpeedArray[Neff]:=SpeedIndex;
        AdvanceRatio := SpeedArray[Neff]/(RPS*Diameter);
        TestEff := PropellerEfficiency;

        If TestEff <= 1.0 then
      EffArray[Neff] := TestEff;

     If TestEff > 1.0 THEN
           EffArray[Neff] := EffArray[Neff-1];

      end;   {while}

     PolySmooth(SpeedArray,EffArray,MaxNeff);

    end;  {if mode=}


   if mode = 1 then
      PropE:=PropEquations;

 end;  {procedure Smooth}


procedure StartPoint;

{Determine Zero Reference Values}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

 procedure PowerToProp;

{Calculate Power input to Prop}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

  var
   Answer  : char;      {Switch}
   Guess   : Real;      {Users power estimate}
   Counter : Integer;

begin
  AdvanceRatio := TrueAirspeed / (RPS * Diameter);
  Answer := 'Y';
  Counter := 0;

  while Answer = 'Y' do
```

```
  begin
   counter := Counter + 1;

   if counter = 1 then
    begin
     Guess := PowerReq / 0.85;
     {WriteLn('ESTIMATED  Power Required INTO Prop: ',
                Guess:3:1);}
    end  {IF}

   else
    begin
     {WriteLn('ACTUAL Power Required  From Prop: ',
                PowerReq:3:1);}
     {WriteLn('Enter Estimated Power into Prop-Disk');}
     ReadLn(Guess)
    end; {ELSE}

   Cp := Guess * ConvHp / (Density * XtoY(RPS, 3) *
         XtoY(Diameter, 5));
   {WriteLn('Advance Ratio is : ', AdvanceRatio : 2 : 2);}
   {WriteLn('Power Coefficient is: ', Cp : 2 : 3);}
   PropBladeAngle := ROUND(BladeAngle);
   {WriteLn('Propeller Blade Angle= ', PropBladeAngle);}
   PropE := PropellerEfficiency;
   {WriteLn('Prop Efficiency:', PropE : 1 : 2);}
   PowerOut := Guess * PropE;
   {WriteLn('ESTIMATED  Power Required From Prop: '
             ,PowerOut:3:1);}
   {WriteLn('ACTUAL Power Required  From Prop: '
             ,PowerReq:3:1);}
   {WriteLn('Guess again? (y or n)');}

   ReadLn(Answer);

   if (Answer = 'Y') or (Answer = 'y') then
    Answer := 'Y'
   else
    Answer := 'N'

  end; {while}

 PowerIn := Guess;

end; {PowerToProp}


procedure ActualHorsepower;

{Get the actual horsepower (BHP) from required horsepower}
{using prop efficiency and auxiliary power.}
```

```
  {COMPLETED: 20 FEB 1989}
  {CHANGE #        DATE              MODIFICATION}

  begin
   BHP := AuxiliaryPower + PowerIn;
   {WriteLn('BHP=', BHP : 3 : 1, ' Altitude=', Altitude);}
   {WriteLn(PRNT, 'BHP=', BHP : 3 : 1, ' Altitude=',
                    Altitude);}
   InitialFuelFlow := FuelPower(1);
   {WriteLn('Initial Fuel Flow from Engine
              Chart:',InitialFuelFlow:2:1);}
  end;  {ActualHorsePower}

 begin
  AirSpeed;
  PowerRequired;
  PowerToProp;
  ActualHorsepower;
 end;     {StartPoint}

 procedure ChangePower;

{Increase or decrease fuel flow for new power setting}
{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE              MODIFICATION}

 begin
  {WriteLn('Enter Throttle Stetting:');}
  {WriteLn('For Accel >',InitialFuelFlow /
             MaxFuelFlow[AltitudeIndex]:1:2);}
  {WriteLn('For Decel  >', 6.2/MaxFuelFlow[AltitudeIndex]
                  :1:2);}

  ReadLn(ThrottleSetting);
  nTests := nTests+1;
  WriteLn('  Throttle =', ThrottleSetting:4:3,'  Tests = '
            ,nTests);
  FlowRate := ThrottleSetting * MaxFuelFlow[AltitudeIndex];

  if InitialFuelFlow < FlowRate then
   Accelerate := TRUE
  else
   Accelerate := FALSE;

  if FlowRate < 1.0 then
   begin
    GLIDE := TRUE;
    PowerAvailable := 0.000;
   end  {IF}

  else
   PowerAvailable := FuelPower(0);
```

```
  {WriteLn('Pav: ', PowerAvailable : 3 : 1);}
 end; {ChangePower}


 procedure FlyPoint;

{Accelerates or decelerates aircraft until velocity change}
{is less than tolerance.  Maintains level flight.  Change}
{in Excess Specific Energy is calculated for use in MCA}
{program.}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

  const
   grav = 32.2;  {ft/sec**2}
   dt = 1;       {Time Increment, 1 second}

  var
   Thrust,Force                : Real;
   WeightIncrement,TotalMass   : Real;
   FLY                         : BOOLEAN;
   Time                        : Integer;

 begin
  {WriteLn('Airspeed= ', TrueAirspeed : 3 : 1);}
  Time := 0;
  FLY := TRUE;
  Cp := PowerAvailable * ConvHp / (Density * XtoY(RPS, 3) *
        XtoY(Diameter, 5));
  WeightIncrement := dt * FlowRate * GalConversion / 3600;

  IF not GLIDE then
   SmoothedPropellerEfficiency(0);

  while FLY do
   begin
    Time := Time + dt;
    ActualWeight := ActualWeight - WeightIncrement;
    TotalMass := ActualWeight / grav;
    PowerRequired;
    AdvanceRatio := TrueAirspeed / (RPS * Diameter);

    if not GLIDE then
     SmoothedPropellerEfficiency(1);

    if PropE < 1.0 then
     begin
      Thrust := (PropE *
              PowerAvailable*ConvHP)/TrueAirspeed;
      Force := Thrust - (ConvHP*PowerReq)/TrueAirspeed;
      dVt := (dt / TotalMass) * Force;
```

```
      end  {IF}

     else
      dVT := 0.0;

     if (ABS(dVt) < Tolerance[AltitudeIndex]) or
           (TrueAirspeed
       < StallSpeed) then
      FLY := FALSE

     else
      begin
       TrueAirspeed := TrueAirspeed + dVt;
       Edot := (TrueAirspeed / grav) * (dVt / dt);
       Write(FlightData,Time,' ',TrueAirspeed:3:1,' '
        ,Edot:2:4,' ',Thrust:3:4);
       WriteLn(FlightData);
      end; {else}

    end; {while}

   Write(FlightData, '999');
   WriteLn(FlightData);
   Write(FlightData, Altitude,' ',FlowRate:2:1,' '
    ,InitialSpeed:3:1);
   WriteLn(FlightData);

  end;  {FlyPoint}


begin  {MCA1 Main Program}
 Proceed := 'y';
 FilePrep;
 InitializeConstants;
 nTests := 0;

 while (Proceed = 'y') or (Proceed = 'Y') do
  begin
   GLIDE := FALSE;
   SetAltitude;
   StartPoint;
   ChangePower;
   FlyPoint;
   {WriteLn('Do You Wish to Continue? (y or n)');}
   ReadLn(Proceed);
  end; {while}

 CLOSE(FlightData);

end.        {MCA1}
```

```
program MCA2(input, output,FlightData);

{Program MCA2 takes data output from MCA1, organizes it by}
{constant airspeed, and sorts the data.  This manipulation}
{is neccesary for use in program MCA3 which is the MCA}
{method of data reduction}


{Coded by Keith Gibby for UTSI Master's Thesis}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE            PROCEDURE MODIFIED}

 type
  CURVE  = array[0..140] of real;
  MATRIX = array[1..60, 1..60] of Real;


 var
  Airspeed, Thrust          : CURVE;   {Actual Thrust lb}
  Edot                      : CURVE;   {Excess Specific}
                                       {Energy (ft/sec)}

  NewThrust,NewEdot         : MATRIX;  {Interpolated}
  FlowRate                  : MATRIX;  {Fuel Flow}

  FuelFlow                  : real;

  Time,MaxTime,MaxTests     : Integer;
  SpeedTic,SpeedIndex       : Integer; {Integer Values}
  MaxIndex,Altitude         : Integer;
  Mode                      : Integer; {Dataset number}

  FlightData, MCAInputData  : TEXT;    {Input/Output Files}

 procedure FileReadPrep;

{Open flight data file from MCA1 program}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE            MODIFICATION}

 begin
  OPEN(FlightData, 'MCA1.OUT',history := OLD);
  RESET(FlightData);
 end; {FileReadPrep}


 procedure FileWritePrep;

{Prepare Output File}
```

```
{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE              MODIFICATION}

 begin
  OPEN(MCAInputData, 'MCA2.OUT');
  REWRITE(MCAInputData);
 end; {FileWritePrep}


 procedure ReadData;

{Read Data From MCA1 Program}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE              MODIFICATION}

  var
   HALT : boolean;

 begin
  WriteLn('Reading Data Set');
  HALT := FALSE;

  while not EOF(FlightData) and not HALT do
   begin
    while not EOLN(FlightData) and not HALT do
     begin
      while not HALT do
       begin
        read(FlightData, Time);

        if (Time = 0) or (Time > 140) then
         HALT := TRUE;

        if (Time > 0) and (Time < 140) and not HALT then
         begin
          Read(FlightData, Airspeed[Time]);
          Read(FlightData, Edot[Time]);
          MaxTime := Time;
          Read(FlightData, Thrust[Time]);
          ReadLn(FlightData);
         end {end if1}

        else
         begin
          read(Flightdata, Altitude);
          read(FlightData, FuelFlow);
          read(FlightData, Airspeed[0]);
         end; {if/else}

       end;  {while Halt}
     end; {while EOLN}
```

```
    ReadLn(FlightData);

   end;{while EOF}

  WriteLn('Data Set Read');

 end; {ReadData}


 procedure SmoothData;

{This procedure picks six points around a constant cardinal}
{airspeed and interpolates for a value at the airspeed.}
{The data are collected in a matrix and are sorted in}
{ascending order by fuel flow at the constant airspeed.}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE            PROCEDURE MODIFIED}

  const
   Npoints = 6;                          {Interpolation Window}

  type
   Packet  = array[1..60] of Real;       {Data Type}

  var
   SpeedX, EdotY,ThrustY : Packet;       {Data Sample Values}
   Delta                 : CURVE;        {Data Window Size}

   StartTime, EndTime    : Integer;
   Values, Count         : Integer;      {Counters}

   InRange : boolean;                    {Switch}


  function Interpolate (xa, ya : Packet;
          k : integer;
          x : Integer) : Real;

{Interpolation Routine.  xa,ya are input array, k is number}
{of data points, x is the point of interest}

{Ref: "NUMERICAL RECIPES", Press, Flannery, Teulolsky, and}
{Vetterling}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}

   var
    c, d     : Packet;
    w, hp, ho, dift, dif, den, dy, y : Real;
    ns, m, i : integer;
```

```
begin
 ns := 1;
 dif := ABS(x - xa[1]);

 for i := 1 to k do
  begin
   dift := ABS(x - xa[i]);

   if (dift < dif) then
    begin
     ns := i;
     dif := dift
    end;

   c[i] := ya[i];
   d[i] := ya[i];

  end;

 y := ya[ns];
 ns := ns - 1;

 for m := 1 to k - 1 do
  begin
   for i := 1 to k - m do
    begin
     ho := xa[i] - x;
     hp := xa[i + m] - x;
     w := c[i + 1] - d[i];
     den := ho - hp;

     if (den = 0.0) then
      begin
       writeln('Pause in Interpolator Routine');
       readln
      end;

     den := w / den;
     d[i] := hp * den;
     c[i] := ho * den

    end;

   if ((2 * ns) < (k - m)) then
    begin
     dy := c[ns + 1]
    end

   else
    begin
     dy := d[ns];
     ns := ns - 1
```

```
      end;

     y := y + dy
    end;

   Interpolate := y;

  end;  {function Interpolate}


  procedure BuildDeltas;

{This procedure finds data, that are within the airspeed}
{window, and time tags the values for retrieval by GetData.}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE           MODIFICATION}

   const
    Window = 20; {For values within 20 fps of airspeed
point}

   var
    STOP : BOOLEAN;   {Switch}

  begin
   STOP   := FALSE;
   Time   := 1;
   Values := 0;

   while not STOP do
    begin
     if ABS(SpeedTic - Airspeed[Time]) < Window then
      begin

       Delta[Time] := ABS(SpeedTic - Airspeed[Time]);
       Values := Values + 1;

       if Values = 1 then
        StartTime := Time;

       if Values > 1 then
        EndTime := Time;

      end;  {if ABS...}

     Time := Time + 1;

     if Time > MaxTime then
      STOP := TRUE;
    end;  {While}
```

```
  end; {BuildDeltas}


  procedure GetData;

{Time tagged values are put into a matrix for use in the}
{interpolating routine.}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE            MODIFICATION}

   var
    HoldIndex, TimeIndex    : Integer;
    Finished                : BOOLEAN;

  begin
   Finished := FALSE;
   Count    := 0;

   while not Finished do
    begin
     HoldIndex := StartTime;

     for TimeIndex := StartTime to (EndTime - 1) do
       if Delta[HoldIndex] > Delta[TimeIndex + 1] then
        HoldIndex := TimeIndex + 1;

     Delta[HoldIndex] := 100.0;
     Count := Count + 1;
     SpeedX[Count] := Airspeed[HoldIndex];
     EdotY[Count] := Edot[HoldIndex];
     ThrustY[Count] := Thrust[HoldIndex];

     if (Count = Npoints) or (count = EndTime) then
      Finished := TRUE;

    end;  {while}

  end;  {GetData}


  procedure FormatData;

{Collected data is interpolated and put into a new array}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE            MODIFICATION}

  begin
   NewEdot[Speedindex, Mode] := Interpolate(SpeedX, EdotY,
                                count, SpeedTic);
```

```
    NewThrust[Speedindex,Mode] := Interpolate(SpeedX,ThrustY,
                                              count,SpeedTic);

    FlowRate[Speedindex, Mode] := FuelFlow;

  end;  {FormatData}

begin  {Start Procedure SmoothData}
  SpeedTic := (SpeedIndex - 1) * 5 + 90;
  MaxIndex := MaxIndex + 1;
  InRange := FALSE;
  BuildDeltas;

  if (Airspeed[1] < Airspeed[MaxTime]) and ((SpeedTic >
                  Airspeed[1]) then
  if (SpeedTic < AirSpeed[MaxTime])) then
    begin
     InRange := TRUE;
     GetData;
     FormatData;
    end;   {if}

  if (Airspeed[0] > Airspeed[MaxTime]) and ((SpeedTic <
                  Airspeed[0]) then
  if (SpeedTic > AirSpeed[MaxTime])) then
    begin
     InRange := TRUE;
     GetData;
     FormatData;
    end; {if}

  if not InRange then
   begin
    NewEdot[Speedindex, Mode]    := 0.0;
    NewThrust[Speedindex, Mode] := 0.0;
    FlowRate[Speedindex, Mode]   := 0.0;
   end

  else
   MaxIndex := MaxIndex + 1;

end;  {SmoothData}


procedure SortData;

{Interpolated data is sorted into ascending order for each}
{airspeed by fuel flow rate}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE           MODIFICATION}
```

```
  var
   i, j, k, check : integer;

  procedure ChangeDataIndex;

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE              MODIFICATION}

  begin
   NewEdot[i, Maxtests + 1] := NewEdot[i, j + 1];
   NewEdot[i, j + 1] := NewEdot[i, j];
   NewEdot[i, j] := NewEdot[i, Maxtests + 1];

   NewThrust[i, Maxtests + 1] := NewThrust[i, j + 1];
   NewThrust[i, j + 1] := NewThrust[i, j];
   NewThrust[i, j] := NewThrust[i, Maxtests + 1];

   FlowRate[i, Maxtests + 1] := FlowRate[i, j + 1];
   FlowRate[i, j + 1] := FlowRate[i, j];
   FlowRate[i, j] := FlowRate[i, Maxtests + 1];

  end; {ChangeDataIndex}

 begin  {Start procedure SortData}
  WriteLn('Sorting Data');

  for i := 1 to 24 do
   begin
    WriteLn('Speed=', (i - 1) * 5 + 90);
    check := 0;
    while check < MaxTests do
     begin

      for j := 1 to MaxTests - 1 do

       if FlowRate[i, j] >= FlowRate[i, j + 1] then
        ChangeDataIndex;

      for k := 1 to MaxTests - 1 do

       if FlowRate[i, k] <= FlowRate[i, k + 1] then
        Check := Check + 1

       else
        Check := 0;

     end; {while}

   end;{for 24}

 end;  {SortData}
```

```
 procedure WriteData;

{Data is written to 'MCA2.OUT' for input to program MCA3}

{COMPLETED: 9 FEB 1990}
{CHANGE #        DATE            MODIFICATION}

  var
   i, j : integer;
   WriteAltitude : Boolean;

 begin
  WriTeln('Outputting Sorted Data');

  for i := 1 to 24 do
   begin
    WriteAltitude := FALSE;

    for j := 1 to MaxTests do

     if FlowRate[i, j] > 0.005 then
      begin
       Write(MCAInputData,i,' ',(i-1)*5+90,' ',FlowRate[i,
             j] : 2 : 2);

       Write(MCAInputData, ' ', NewEdot[i, j]:2:2);
       Write(MCAInputData, ' ', NewThrust[i, j]:4:2);
       WriteLn(MCAInputData);
       WriteAltitude := TRUE;
      end;{if/for ..MaxTests}

    if WriteAltitude then
     WriteLn(MCAInputData, Altitude);
   end;  {for .. 24}

  WriteLn(MCAInputData, '222');

 end; {WriteData}


begin   {MCA2 Main Program}
 FileReadPrep;
 FileWritePrep;
 Mode := 0;
 WriteLn('Enter Number Of Max Tests : ');
 ReadLn(MaxTests);

 while not EOF(FlightData) do
  begin
   ReadData;
   mode := mode + 1;
```

```
    Write('Altitude=', Altitude,' #Points ',MaxTime,'
          ','Fuel= ',FuelFlow:2:1);

    WriteLn;

    for SpeedIndex := 1 to 24 do
     SmoothData;

   end;{while}

  WriteLn('Data Formatting Complete');
  SortData;            {Data is sorted by constant airspeed}
  WriteData;           {For MCA3}

  CLOSE(MCAInputData);
  CLOSE(FlightData);  {Data is ready for MCA3}

end.  {Program MCA2}
```

```
program MCA3(input, output,MCAInputData);

{This program uses constant airspeed curves from MCA2 to}
{generate Thrust and Drag using the MCA method.  Data are}
{read from MCA2.OUT and Reference Points are chosen and}
{used to generate Efficiency curves.  At each reference}
{point the efficiency curve reaches a maximum. Thrust and}
{Drag are calculated at each reference point and then}
{compared to the model values from MCA1.}


{Coded by Keith Gibby for UTSI Master's Thesis}

{COMPLETED: 20 FEB 1990}
{CHANGE #        DATE              PROCEDURE MODIFIED}


 const
  FuelHeatingValue = 19300;
  BTUconv          = 778;
  HourToSec        = 3600;
  GaltoLbs         = 6.5;
  PowerTerm        = 0.0;
  SLdensity        = 0.0023769;
  ActualWeight     = 2400;
  StandardWeight   = 2650;

 type
  CURVE = array[1..140] of real;   {Data Type}

 var
  SpeedIndex, SpeedTic, Altitude,Min : integer;
  CurvePoints,Npoints,MaxEXP,HoldExp : integer; {Counters}

  FuelFlow,MassFlow          : CURVE;
  Edot,Thrust, Efficiency    : CURVE;
  NEWEdot,NEWThrust          : CURVE;
  MCAThrust, MCADrag         : CURVE;
  ThrustDelta,DragDelta      : CURVE;
  ThrustPCerror,DragPCerror  : CURVE;
  dEdFStar                   : CURVE;

  FuelFlowRef,Power,FuelFlowStar : Real;
  EdotRef, EdotStar,dEdF         : Real;
  dEdF, UnitsFactor,Drag,Weight  : Real;
  MinDrag                        : Real;

  HOLDcoeff,EdotCoeffs,ThrustCoeffs : array [1..7] of real;

  STOP : boolean;

  MCAInputData, MCAOutputData,Results : TEXT;
```

```
 procedure InputFilePrep;
 begin
  OPEN(MCAInputData, 'MCA2.OUT',history:=OLD);
  RESET(MCAInputData);
 end; {InputFilePrep}

 procedure OutputFilePrep;
 begin

  OPEN(Results, 'Results.mca',history:=NEW);
  REWRITE(Results);

  OPEN(MCAOutputData, 'MCA3.OUT',history:=NEW);
  REWRITE(MCAOutputData);

 end; {OutputFilePrep}

 procedure UnitConversions;
 begin
  UnitsFactor := HourToSec / (GalToLbs * BTUconv);
 end;

 procedure ReadSortedData;

{COMPLETED: 20 FEB 1990}
{CHANGE #        DATE              MODIFICATION}

  var
   i : integer;
   HALT : Boolean;
 begin
   HALT := FALSE;
  i := 0;
  while not EOF(MCAInputData) and not HALT do
   begin
    while not HALT do
     begin
      read(MCAInputData, SpeedIndex);
      if (SpeedIndex < 1) or (SpeedIndex > 100) then
       HALT := TRUE;
      if (SpeedIndex = 222) then
       STOP := TRUE;
      if (SpeedIndex > 0) and not HALT then
       begin
        i := i + 1;
        Npoints := i;

Read(MCAInputData,SpeedTic,FuelFlow[i],Edot[i],Thrust[i]);

        ReadLn(MCAInputData);
       end; {if}
     end; {while ...EOLN}
```

```
   end;  {while...EOF}
 end; {end Read}


function XtoY (Base, Exponent : Real) : Real;

{Math tool to raise base X to power Y}
{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE            MODIFICATION}


 begin
  XtoY := exp(Exponent * Ln(Base));
 end;  {XtoY}


FUNCTION PolySmooth(XArray,YArray:CURVE;
                    NumDataPairs:integer;
                    x : Real;Degree:integer):Real;

{Polynomial Curve Fit Routine.}

{Adapted from "TURBO PASCAL PROGRAM LIBRARY", by Rugg and}
{Feldman}


const
   RowSize   = 250;
   ColSize   = 2;
   MaxDegree = 7;

type
   CoeffArrayType = array[0..30] of real;

var
   PowerArray, RHS, Coeffs : CoeffArrayType;
   Factors                 : array[1..15, 1..15] of real;
   TwoDegree, NumEqns, Code : integer;
   J, K, L, M, Last                                  : integer;
   CharFlag                                          : char;
   TimeToQuit, NoMoreToDo                            : boolean;
   Work, Numer, Denom, Correlation, MeanY, Y    : real;

procedure pInterpolate;

begin
  Y := 0.0;
  for K := 1 to NumEqns do
    Y := Y + Coeffs[K] * XtoY(X, K - 1);
end;

procedure GetCoefficients;
```

```
{Coefficients to be used}

begin
 MaxEXP := NumEqns;
 for K:= 1 TO Numeqns do
  HOLDcoeff[K] := Coeffs[k];
end;  {Get..}

procedure SolveEqns;

begin
   TwoDegree := Degree * 2;
   NumEqns   := Degree + 1;
   for J := 0 to TwoDegree do
      begin
         PowerArray[J] := 0.0;
         for K := 1 to NumDataPairs do
          PowerArray[J] := PowerArray[J]+XtoY(XArray[K], J)
      end;
   for J := 1 to NumEqns do
      begin
         RHS[J] := 0.0;
         for K := 1 to NumDataPairs do
            RHS[J] := RHS[J] + YArray[K] *
                              XtoY(XArray[K], J - 1)
      end;
   for J := 1 to NumEqns do
      for K := 1 to NumEqns do
         Factors[J,K] := PowerArray[J + K - 2];
   for K := 1 to Degree do
      begin
         M := K + 1;
         L := K;
         repeat
            if abs(Factors[M,K]) > abs(Factors[L,K]) then
               L := M;
            M := M + 1
         until
            M > NumEqns;
         for J := K to NumEqns do
            begin
               Work          := Factors[K,J];
               Factors[K,J] := Factors[L,J];
               Factors[L,J] := Work;
            end;
         Work := RHS[K];  RHS[K] := RHS[L];  RHS[L] := Work;
         M := K + 1;
         repeat
            Work          := Factors[M,K] / Factors[K,K];
            Factors[M,K] := 0.0;
            for J := K + 1 to NumEqns do
               Factors[M,J] := Factors[M,J] -
```

```
                                    Work * Factors[K,J];
               RHS[M] := RHS[M] - Work * RHS[K];
               M      := M + 1
            until
               M > NumEqns
         end;
      Coeffs[NumEqns] := RHS[NumEqns] /
Factors[NumEqns,NumEqns];
      for M := Degree downto 1 do
         begin
            Work := 0.0;
            for J := M + 1 to NumEqns do
               begin
                  Work      := Work + Factors[M,J] * Coeffs[J];
                  Coeffs[M] := (RHS[M] - Work) / Factors[M,M]
               end
         end;
      Numer := 0.0;
      Denom := 0.0;
      for J := 1 to NumDataPairs do
         begin
            Work := 0.0;
            for K := 1 to NumEqns do
               Work := Work+Coeffs[K] * XtoY(XArray[J], K - 1);
            Numer := Numer + sqr(YArray[J] - Work);
            Denom := Denom + sqr(YArray[J] - MeanY)
         end;
      if Denom = 0.0 then
         Correlation := 1.0
      else
         Correlation := sqrt(1.0 - Numer / Denom);
end;

BEGIN
   SolveEqns;
   pInterpolate;
   GetCoefficients;
   PolySmooth := Y;
END;   {PolySmooth}


function derivative(X:real):real;

{Takes derivative of polynomial function}
{Must have Polynomial coefficients from PolySmooth}

{COMPLETED: 20 FEB 1990}
{CHANGE #        DATE            MODIFICATION}

var
 dYdX : Real;
 p : integer;
```

```
begin
  dYdX := 0;
  for p:= 2 to HoldEXP do
   dYdX := dYdX + (p-1)*EdotCoeffs[p]*XtoY(X,(p-2));
  Derivative := dYdX;
end;  {derivative}


 function PowerReq : REAL;

{Determine actual power required from prop by correcting}
{Standard Day Power Required vs Velocity Charts for weight}
{and altitude.}

{COMPLETED: 20 FEB 1989}
{CHANGE #        DATE              MODIFICATION}

  const
   A0 = 136.0536855;
   A1 = -1.375733333;
   A2 = 6.06197163e-3;
   DensityRatio = 1;   {for sea-level condition}
  var
   WeightRatio,Vew,THP : REAL;

 begin
  WeightRatio := StandardWeight / Weight;
  Vew := SpeedTic * SQRT(DensityRatio) * SQRT(WeightRatio);
  THP := A0 + A1 * Vew + A2 * SQR(Vew);
  PowerReq := THP / (SQRT(DensityRatio)
                  *exp(1.5*Ln(WeightRatio)));
 end;  {PowerReq}

procedure BuildCurve;

{This procedure builds a fuel flow array from Min to max}
{fuel flow}

{COMPLETED: 20 FEB 1990}
{CHANGE #        DATE              MODIFICATION}


 const
  FlowIncrement = 0.1000;

 var
  newJ : integer;
  FlowSum : real;

 begin
  newJ := 0;
  FlowSum := FuelFlow[1] - FlowIncrement;
```

```
  while (FlowSum <= FuelFlow[Npoints]) do
   begin
    newJ := newJ + 1;
    FlowSum := FlowSum + FlowIncrement;
    Massflow[newJ] := FlowSum;
   end; {while}

  CurvePoints := newJ;

 end; {BuildCurve}


procedure CalculateCurves;

{Using coefficients from the polynomial fit the functions}
{in this procedure are used to calculate Edot or Thrust}

{COMPLETED: 20 FEB 1990}
{CHANGE #        DATE              MODIFICATION}


var
  MaxEfficiency : REAL;
  RefIndex,j, MaxJ : Integer;


  function EdotEquation(Flow:real;Fit:Integer) : real;

   var
    z : integer;
    y : real;

   begin
    y := 0;

    for z := 1 to (Fit + 1) do
     y := y + EdotCoeffs[z]* XtoY(Flow,z-1);

    EdotEquation := y;

   end;   {EdotEquation}

  function ThrustEquation(Flow:real;Fit:integer) : real;

   var
    z : integer;
    y : real;

   begin
    y := 0;

    for z := 1 to (Fit + 1) do
```

```
      y := y + ThrustCoeffs[z]* XtoY(Flow,z-1);

    ThrustEquation := y;

   end;    {ThrustEquation}


 procedure GetReferencePoint;

{Get Reference Points from data array}

{COMPLETED: 20 FEB 1990}
{CHANGE #         DATE              MODIFICATION}


 var
  m : integer;

  begin
     FuelFlowRef := MassFlow[RefIndex];

     if RefIndex = 1 then
      begin
       NewThrust[1] :=
          PolySmooth(FuelFlow,Thrust,Npoints,FuelFlowRef,2);
       for m := 1 to MaxExp do
        ThrustCoeffs[m] := HOLDCoeff[m];
       EdotRef :=
          PolySmooth(FuelFlow,Edot,Npoints,FuelFlowRef,3);
       for m := 1 to MaxEXP do
        EdotCoeffs[m] := HOLDCoeff[m];
       HoldEXp := MaxExp
      end
     else
      EdotRef := EdotEquation(FuelFlowRef,3);

     dEdF := Derivative(FuelFlowRef);
     FuelFlowStar := FuelFlowRef;
     EdotStar := EdotRef;
     dEdFSTAR[RefIndex] := dEdF;
  end; {GetRef...}

  procedure GenerateEfficiencyCurves;

{Calculates the efficiency from min to max fuel flow}

{COMPLETED: 20 FEB 1990}
{CHANGE #         DATE              MODIFICATION}

   var
    A, B : Real;
```

```
  begin
   MaxEfficiency := -999;
   for J := 1 to CurvePoints do
    begin
     A := UnitsFactor * Weight / MassFlow[j];
     A := A / FuelHeatingValue;
     NEWEdot[j] := EdotEquation(MassFlow[j],3);
     NEWThrust[j] := ThrustEquation(MassFlow[j],2);
     B := NEWEdot[j] - EdotRef + FuelFlowRef * dEdf;
     {Units Cancel}
     Efficiency[j] := A * B;

    end;  {FOR}


  end; {Gen...Curve}

  procedure CalculateThrustAndDrag;

{Calculate thrust and drag at reference points}

{COMPLETED: 20 FEB 1990}
{CHANGE #        DATE              MODIFICATION}

  var
   Term1   : real;
   R : Integer;

  begin
   for j := 1 to CurvePoints do
    begin
     {Calculate all thrust values for Fuel Flows}
     TERM1 := NEWEdot[j] - EdotSTAR + FuelFlowSTAR *
           dEdfSTAR[j];

     If j = RefIndex then  {Reference Point values}
      begin
      MCAThrust[j] := (Term1 * Weight / SpeedTic) -
                   PowerTerm;
      ThrustDelta[j] := MCAThrust[j]-NewThrust[j];
      {Compare to Model}
      MCADrag[j] := (MCAThrust[j] - Weight * EdotSTAR /
                 SpeedTic);
      DragDelta[j] := MCADrag[j] - (Power * 550) /
                   SpeedTic;

      if Newthrust[j] <> 0.0 then
       ThrustPCerror[j] := 100*ThrustDelta[j]/NewThrust[j];
      DragPCerror[j] := 100 * DragDelta[j] /
                     (Power*550.0/SpeedTic);

       If RefIndex = 1 then
```

```
        MinDrag := ABS(DragPCerror[RefIndex])
       else
        if Mindrag < ABS(DragPCerror[RefIndex]) then
         begin
        MinDrag := Mindrag;
           Min := Min;
         end
        else
         begin
           Mindrag := ABS(DragPCerror[RefIndex]);
           Min := RefIndex
         end;

       R := RefIndex;
       Write(MCAOutputData,'          ',FuelFlowStar:6:1);
       Write(MCAOutputData,'      ',EdotStar:6:2,'       ',
            dedfstar[r]:6:3);
       Write(MCAOutputData,'       ',ThrustPCerror[R]:7:3);
       WriteLn(MCAOutputData,'       ',DragPCerror[R]:7:3);

     end; {if..}
   end;    {for..}

 end; {Calc...Thrust}


begin
 for RefIndex := 1 to CurvePoints do
  begin
   GetReferencePoint;
   GenerateEfficiencyCurves;
   CalculateThrustAndDrag;
  end;  {FOR...}
 Write(Results,SpeedTic:4,' ',MASSFlow[Min]:5:1,' ',
       NewEdot[Min]:7:2);
 write(Results,' ',dEdFStar[min]:7:4);
 WriteLn(Results,'    ',ThrustPCerror[Min]:6:1,'  ',
       DragPCerror[Min]:6:1);
 WriteLn(MCAOutputData);
end; {Calculate...}


begin  {MCA3 Main Program}

 InputFilePrep;
 OutPutFilePrep;
 UnitConversions;
 STOP := FALSE;
 Weight := ActualWeight;
 MIN := 1;
 while not STOP do
  begin
```

```
    ReadSortedData;
    BuildCurve;
    Power := PowerReq;
    if not STOP then
     begin
      WriteLn(' Speed= ',SpeedTic);
      Write(MCAOutputData,'          ','Fuel Flow');
      Write(MCAOutputData,'     Edot','          Slope');
      Write(MCAOutputData,'     Thrust Error);
      WriteLn(MCAOutputData,'   Drag Error');

      CalculateCurves;
    end; {if}

   end; {while}

  CLOSE(MCAInputData);
  CLOSE(MCAOutputData);
  CLOSE(Results);

end. {MCA3}
```

APPENDIX 3

Appendix 3 contains Tables 1 through 4 for this study.

Table 1

Thrust and Drag Errors at Reference Points, 100 fps

Fuel Flow	Excess Specific Energy	Slope	% Thrust Error	% Drag Error
6.0	-1.94	4.045	147.547	127.155
6.1	-1.54	3.938	137.357	121.534
6.2	-1.16	3.832	127.778	115.918
6.3	-0.78	3.729	118.750	110.308
6.4	-0.41	3.627	110.223	104.708
6.5	-0.05	3.527	102.150	99.122
6.6	0.29	3.429	94.494	93.551
6.7	0.63	3.333	87.219	88.000
6.8	0.96	3.238	80.296	82.472
6.9	1.28	3.145	73.697	76.969
7.0	1.59	3.055	67.399	71.496
7.1	1.89	2.965	61.381	66.054
7.2	2.18	2.878	55.623	60.647
7.3	2.47	2.793	50.109	55.279
7.4	2.74	2.709	44.825	49.952
7.5	3.01	2.627	39.755	44.669
7.6	3.27	2.547	34.888	39.435
7.7	3.52	2.469	30.213	34.251
7.8	3.76	2.393	25.720	29.121
7.9	4.00	2.318	21.400	24.048
8.0	4.23	2.245	17.245	19.036
8.1	4.45	2.174	13.248	14.087
8.2	4.66	2.105	9.401	9.204
8.3	4.87	2.038	5.699	4.392
8.4	5.07	1.972	2.136	-0.348
8.5	5.26	1.909	-1.293	-5.012
8.6	5.45	1.847	-4.591	-9.597
8.7	5.63	1.787	-7.765	-14.099
8.8	5.81	1.728	-10.816	-18.516
8.9	5.98	1.672	-13.749	-22.845
9.0	6.14	1.617	-16.567	-27.082
9.1	6.30	1.565	-19.271	-31.224
9.2	6.45	1.514	-21.866	-35.268
9.3	6.60	1.464	-24.352	-39.211
9.4	6.75	1.417	-26.733	-43.049
9.5	6.89	1.371	-29.009	-46.780
9.6	7.02	1.328	-31.182	-50.401
9.7	7.15	1.286	-33.253	-53.908
9.8	7.28	1.246	-35.225	-57.298
9.9	7.40	1.207	-37.096	-60.568
10.0	7.52	1.171	-38.870	-63.715
10.1	7.64	1.136	-40.545	-66.736

Table 2

Thrust and Drag Errors at Reference Points, 150 fps

Fuel Flow	Excess Specific Energy	Slope	% Thrust Error	% Drag Error
6.5	-1.35	4.626	144.934	127.854
6.6	-0.89	4.510	135.084	122.340
6.7	-0.45	4.396	125.836	116.848
6.8	-0.01	4.284	117.135	111.384
6.9	0.41	4.175	108.931	105.949
7.0	0.82	4.068	101.181	100.547
7.1	1.22	3.963	93.848	95.182
7.2	1.61	3.860	86.899	89.856
7.3	1.99	3.760	80.305	84.574
7.4	2.37	3.662	74.039	79.337
7.5	2.73	3.565	68.080	74.150
7.6	3.08	3.472	62.405	69.016
7.7	3.42	3.380	56.998	63.937
7.8	3.76	3.291	51.841	58.918
7.9	4.08	3.204	46.919	53.961
8.0	4.40	3.119	42.221	49.069
8.1	4.70	3.036	37.733	44.246
8.2	5.00	2.956	33.444	39.496
8.3	5.29	2.878	29.346	34.820
8.4	5.58	2.802	25.429	30.223
8.5	5.86	2.728	21.686	25.708
8.6	6.12	2.657	18.108	21.278
8.7	6.39	2.588	14.690	16.936
8.8	6.64	2.521	11.426	12.686
8.9	6.89	2.456	8.309	8.530
9.0	7.13	2.393	5.336	4.472
9.1	7.37	2.333	2.502	0.516
9.2	7.60	2.275	-0.197	-3.336
9.3	7.82	2.219	-2.765	-7.080
9.4	8.04	2.166	-5.206	-10.714
9.5	8.26	2.114	-7.521	-14.233
9.6	8.47	2.065	-9.714	-17.635
9.7	8.67	2.018	-11.787	-20.916
9.8	8.87	1.974	-13.742	-24.074
9.9	9.07	1.931	-15.581	-27.104
10.0	9.26	1.891	-17.306	-30.004
10.1	9.44	1.853	-18.919	-32.770
10.2	9.63	1.817	-20.419	-35.400
10.3	9.81	1.784	-21.810	-37.889
10.4	9.98	1.752	-23.090	-40.235
10.5	10.16	1.723	-24.262	-42.435
10.6	10.33	1.697	-25.326	-44.485

Table 3

Thrust and Drag Errors at Reference Points, 200 fps

Fuel Flow	Excess Specific Energy	Slope	% Thrust Error	% Drag Error
7.2	-6.52	4.577	110.021	74.292
7.3	-6.07	4.447	101.943	70.132
7.4	-5.63	4.320	94.284	66.018
7.5	-5.20	4.197	87.016	61.954
7.6	-4.79	4.076	80.114	57.942
7.7	-4.39	3.959	73.553	53.985
7.8	-4.00	3.845	67.312	50.086
7.9	-3.62	3.735	61.374	46.248
8.0	-3.25	3.627	55.722	42.474
8.1	-2.89	3.523	50.340	38.766
8.2	-2.55	3.422	45.215	35.127
8.3	-2.21	3.324	40.334	31.560
8.4	-1.88	3.229	35.687	28.069
8.5	-1.56	3.137	31.264	24.655
8.6	-1.25	3.049	27.055	21.321
8.7	-0.95	2.964	23.053	18.072
8.8	-0.66	2.882	19.249	14.908
8.9	-0.38	2.804	15.638	11.834
9.0	-0.10	2.728	12.213	8.851
9.1	0.17	2.656	8.968	5.963
9.2	0.43	2.587	5.899	3.173
9.3	0.69	2.521	3.001	0.483
9.4	0.93	2.458	0.270	-2.103
9.5	1.18	2.399	-2.297	-4.583
9.6	1.41	2.342	-4.705	-6.955
9.7	1.65	2.289	-6.955	-9.215
9.8	1.87	2.240	-9.051	-11.360
9.9	2.09	2.193	-10.995	-13.388
10.0	2.31	2.150	-12.788	-15.295
10.1	2.52	2.109	-14.433	-17.080
10.2	2.73	2.072	-15.931	-18.738
10.3	2.94	2.039	-17.284	-20.268
10.4	3.14	2.008	-18.493	-21.666
10.5	3.34	1.981	-19.557	-22.930
10.6	3.54	1.956	-20.480	-24.057
10.7	3.73	1.935	-21.260	-25.043
10.8	3.92	1.918	-21.898	-25.887
10.9	4.12	1.903	-22.394	-26.585
11.0	4.30	1.892	-22.749	-27.134
11.1	4.49	1.883	-22.962	-27.532
11.2	4.68	1.879	-23.033	-27.776

TABLE 4

Minimum Errors Across Speed Range

V	$\dot{m}$	$\dot{e}$	$d\dot{e}/d\dot{m}$	%ΔT	%ΔD
90	8.2	3.71	1.9304	3.0	2.0
95	8.3	4.39	1.9313	2.2	-0.3
100	8.4	5.07	1.9724	2.1	-0.3
105	8.5	5.69	2.0084	1.8	-0.7
110	8.6	6.24	2.0381	1.3	-1.5
115	8.6	6.52	2.1281	3.9	2.3
120	8.7	6.93	2.1503	3.1	1.1
125	8.8	7.27	2.1690	2.2	-0.2
130	8.9	7.53	2.1810	1.1	-1.8
135	8.9	7.50	2.2534	3.1	1.0
140	9.0	7.62	2.2544	1.6	-1.1
145	9.0	7.42	2.3181	3.2	1.3
150	9.1	7.37	2.3330	2.5	0.5
155	9.2	7.25	2.3304	1.1	-1.3
160	9.2	6.81	2.3737	2.0	0.0
165	9.3	6.51	2.3702	0.9	-1.4
170	9.3	5.89	2.4306	2.8	1.1
175	9.3	5.20	2.4481	2.9	1.0
180	9.3	4.45	2.4485	2.3	0.0
185	9.2	3.41	2.4853	3.6	0.8
190	9.3	2.70	2.4895	2.9	0.5
195	9.3	1.75	2.4766	1.9	-0.9
200	9.3	0.69	2.5208	3.0	0.5
205	9.4	-0.20	2.5078	0.8	-0.5

VITA

John K. Gibby was born in Hayesville, North Carolina on November 23, 1958, the son of Sherman W. Gibby and Evelyn M. Gibby. He attended elementary through high school at Hayesville High School, and graduated from there in 1977. In 1981 he received a Bachelor of Science Degree in Aerospace Engineering from North Carolina State University, and was commissioned as an officer in the United States Air Force. His first active duty assignment was at NASA Ames Research Center. Then, in 1984, he was reassigned to Arnold Engineering Development Center in Tullahoma, Tennessee. From there, in 1987, he was assigned to his present location in Las Vegas, Nevada.

Captain Gibby is married to the former Miss Norma R. Stanley of Greensboro, North Carolina. Their two children are Megan and Laura.