AIAA-2002-4704

System IDentification Programs for AirCraft (SIDPAC)

Eugene A. Morelli
NASA Langley Research Center
Hampton, VA

AIAA Atmospheric Flight Mechanics Conference
August 5-8, 2002 / Monterey, CA

SYSTEM IDENTIFICATION PROGRAMS FOR AIRCRAFT (*SIDPAC*)

Eugene A. Morelli
NASA Langley Research Center
Hampton, Virginia USA 23681 – 2199

Abstract

A collection of computer programs for aircraft system identification is described and demonstrated. The programs, collectively called System IDentification Programs for AirCraft, or *SIDPAC*, were developed in MATLAB® as m-file functions. *SIDPAC* has been used successfully at NASA Langley Research Center with data from many different flight test programs and wind tunnel experiments. *SIDPAC* includes routines for experiment design, data conditioning, data compatibility analysis, model structure determination, equation-error and output-error parameter estimation in both the time and frequency domains, real-time and recursive parameter estimation, low order equivalent system identification, estimated parameter error calculation, linear and nonlinear simulation, plotting, and 3-D visualization. An overview of *SIDPAC* capabilities is provided, along with a demonstration of the use of *SIDPAC* with real flight test data from the NASA Glenn Twin Otter aircraft. The *SIDPAC* software is available without charge to U.S. citizens by request to the author, contingent on the requestor completing a NASA software usage agreement.

Introduction

Developing mathematical models of physical systems based on imperfect observations or measurements is known as *system identification*. When the system to be modeled is an aircraft, the models are generally dynamic with multiple inputs and outputs, and the measurements are noisy. As a result, the required data volume is substantial, and most aircraft system identification tasks require computers and software. Many computer programs for aircraft system identification have been developed over the years, but most have been written in FORTRAN, and have been designed to address a specific type of problem, usually using methods best suited to the application for which the program was developed. The existing programs are scattered among different organizations and people, and many have limited availability. Significant time and effort can be required to learn how to compile and operate a particular program competently. This effort is in addition to the time and effort involved in learning what the program actually does, how the methods are implemented, and how to properly interpret the results.

System IDentification Programs for AirCraft, or *SIDPAC*, is a collection of computer programs that implement a variety of methods for solving problems in the area of system identification applied to aircraft. This paper describes the routines included in *SIDPAC* and demonstrates *SIDPAC* using a real flight test data example. All *SIDPAC* programs were written in MATLAB® as m-file functions. An m-file function in MATLAB® is the analog of a subroutine in FORTRAN or a function in C.

MATLAB® is a platform-independent language that is easy to learn and program. MATLAB® also has many built-in functions for linear algebra, data analysis, plotting, and debugging, and uses double precision arithmetic by default. These characteristics make MATLAB® an excellent choice for implementing *SIDPAC* algorithms.

All of the algorithms implemented in *SIDPAC* are based on information available in the open literature. The documents listed under **References** contain information on the algorithms implemented in *SIDPAC*. The references are grouped according to their relevance to system identification problems addressed in *SIDPAC*.

Development of *SIDPAC* occurred over a period of approximately 10 years at NASA Langley Research Center. The author wrote and tested all of the *SIDPAC* routines. Most of the routines in *SIDPAC* were

* Research Engineer, Senior Member AIAA
e-mail: e.a.morelli@larc.nasa.gov

originally coded as m-file functions, but a few were translated and adapted from FORTRAN codes that have been used for many years. *SIDPAC* has been and continues to be used at NASA Langley Research Center for system identification applied to data from various flight test programs and wind tunnel experiments.

Software tools for specific system identification tasks are implemented in *SIDPAC* as individual m-file functions, which may call other supporting m-file functions. The comment header for each *SIDPAC* routine includes calling syntax, descriptions of inputs and outputs, the creation and modification history, and a list of called *SIDPAC* routines. *SIDPAC* makes use of a few MATLAB® routines from the Signal Processing Toolbox and the Control Systems Toolbox, sold by The MathWorks, Inc., but *SIDPAC* also includes alternate routines that can be used if these MATLAB® toolboxes are not available to the user. There are also a few C mex files included in *SIDPAC* to increase processing speed for certain tasks, but again, there are alternate routines that can be used if a C compiler is not available to the user. *SIDPAC* is therefore a complete set of programs for aircraft system identification, requiring only the standard built-in MATLAB® functions.

SIDPAC was developed and tested under MATLAB® 5.2, release 11, and MATLAB® 6.1, release 12.1, on an IBM-compatible personal computer. *SIDPAC* software will work properly on any computer running MATLAB® 5.2 or higher. Since *SIDPAC* is very much like a MATLAB® toolbox, the computing hardware and operating system requirements for *SIDPAC* are the same as for MATLAB®.

The descriptions and demonstrations included here apply to *SIDPAC* version 1.1, which is the latest version as of this writing. *SIDPAC* version 1.1 software was sent to all registered *SIDPAC* users via e-mail just prior to public release of this paper.

Calling syntax and descriptive material in the header of each *SIDPAC* m-file are displayed in the MATLAB® command window in response to typing `help filename`, where `filename` is the name of a specific m-file. All *SIDPAC* m-files have the customary `.m` file extension. MATLAB® is an interpretive language, so the source code runs directly in the MATLAB® environment, with no requirement for explicit compile and link commands. To execute any m-file in *SIDPAC*, the correct calling syntax can be copied from the comment header directly to the MATLAB® command line, then executed with a carriage return.

The *SIDPAC* m-files were originally written for research purposes and not for public release, so error handling, user interface, and the like, are Spartan. However, the software is fully documented and liberally commented.

The next section gives an overview of the *SIDPAC* software by listing m-files in several system identification problem categories, along with short narrative descriptions of what each m-file does. Following this, example applications of selected routines in *SIDPAC* to real flight test data analysis and modeling problems are presented and discussed.

SIDPAC Overview

Nearly every system identification problem has some aspect that makes it unique. Therefore, it is only a slight exaggeration to say that every system identification problem is a special case. Because of this, *SIDPAC* does not include much in the way of automated analysis or decision-making. Instead, *SIDPAC* implements a wide variety of system identification methods as individual m-files. *SIDPAC* includes Graphical User Interfaces (GUI) that aid the analyst in routine tasks such as unit conversion, signal definition, and data compatibility analysis. Beyond this, the spectrum of choices for inputs and outputs, model forms, transformations, data analysis, and modeling methods is so broad, and the problems are so diverse, that the development of a GUI for these purposes would be either very complicated or not effective. Therefore, after the initial data reduction, *SIDPAC* m-files must be called individually from the MATLAB® command line. This forces the analyst to be cognizant of what methods are being used and how the methods are implemented. This level of contact with the methods and software is essential for obtaining good results.

It is a simple matter to create custom data analysis and modeling scripts in MATLAB® for particular system identification problems, using calls to *SIDPAC* m-files. This is demonstrated in the **Example** section. The resulting script can be used to automate the analysis for different maneuvers, and has the added benefit of completely documenting the data analysis and modeling process.

The m-file functions that comprise the main capabilities of *SIDPAC* are categorized below according to functionality. Short narrative descriptions of the m-files appear below each category heading.

The m-files described here represent only a partial listing of the contents of *SIDPAC*.

Data Analysis

accel_cor.m – Corrects accelerometer measurements from the sensor location to the aircraft c.g.

axcnv.m – Finds vector components in a rotated coordinate system.

bodecmp.m – Compares Bode plots for general transfer function models.

bodeplt.m – Draws Bode plots.

chirpz.m – Computes the chirp-Z discrete Fourier transform.

cmpplt.m – Plots two time series and their difference.

cmpsigs.m – Scales and removes biases from two time series so their waveforms can be compared.

compfc.m – Computes non-dimensional applied force coefficients based on measured data.

compmc.m – Computes non-dimensional applied moment coefficients based on measured data.

compzsd.m – Computes a smoothed numerical time derivative of noisy time series using Fourier analysis and the Wiener filter in the frequency domain.

correl.m – Computes the estimated parameter correlation matrix.

corx.m – Computes the normalized regressor correlation matrix.

csmep.m – Computes smoothed endpoints for a noisy time series using a local cubic least squares fit.

cubic_dtrend.m – Computes a cubic detrend function for a noisy time series.

cutftd.m – Plots flight test data and implements manual cutting of flight test maneuver data lengths.

deriv.m – Computes a smoothed numerical time derivative of noisy time series using local least squares polynomial fits to the data.

dft.m – Computes the discrete Fourier transform using the definition (i.e., without using a Fast Fourier Transform algorithm).

estlag.m – Estimates the pure time delay between two time series, using the maximum slope projection method described in MIL-STD 1797A.

fint.m – Computes a high-accuracy finite Fourier integral for arbitrary frequencies, using measured time series data.

fixdrop.m – Manually fixes data dropouts.

hsmoo.m – Implements *a posteriori* low-pass filtering using fixed weight smoothing.

pwrband.m – Computes the frequency band that contains a given fraction of the power in a time series.

smoo.m – Separates signal from noise for a measured time series, using Fourier analysis and an optimal Wiener filter.

spect.m – Computes power spectral density for a measured time series.

tshift.m – Estimates the relative time shift between two time series, using time-domain cross correlation.

xsmep.m – Computes smoothed endpoints for a measured time series.

Experiment Design

compcrb.m – Computes the Cramér-Rao bounds and the information matrix.

dox.m – Generates modern experiment designs for response surface modeling.

mkfswp.m – Generates a linear or logarithmic frequency sweep input.

mkrandss.m – Generates a sum of sine waves with random amplitudes and frequencies.

mkrdn.m – Generates white or arbitrarily colored random noise inputs.

mksqw.m – Generates an alternating square wave input with arbitrary amplitudes and pulse widths.

mksswp.m – Generates a Schroeder sweep input.

Data Compatibility Analysis

airchk.m – Checks the compatibility of measured data for translational accelerations, body-axis angular rates, airspeed, sideslip angle, and angle of attack.

compat.m – Estimates instrumentation error parameters from measured input-output time-domain data, using output-error maximum likelihood.

dcmp.m – Integrates aircraft kinematic equations for data compatibility analysis.

sens_cor.m – Applies instrumentation error corrections to measured data.

rotchk.m – Checks the compatibility of measured data for body-axis angular rates and Euler angles.

Model Structure Determination

gsorth.m – Generates a set of mutually orthogonal regressors using Gram-Schmidt orthogonalization.

swr.m – Identifies general multivariate models from measured input-output data using least squares stepwise regression, and computes parameter estimates and error bounds. This routine requires interactive input from the analyst to identify the model structure. The routine can handle either real or complex data, so the same routine can be used for time-domain or frequency-domain data.

offit.m – Identifies general multivariate models from measured input-output time-domain data, and computes parameter estimates and error bounds. Multivariate orthogonal basis functions generated from the measured data are used to identify the model structure. This routine can be run autonomously or with interactive input from the analyst.

pfstat.m – Computes the partial F statistic for hypothesis testing in model structure determination.

Equation-Error Parameter Estimation

lesq.m – Estimates equation-error model parameters using a linear least squares regression formulation. This routine can be used with either real or complex data, so the same routine can be used for time-domain or frequency-domain data.

Output-Error Parameter Estimation

mnr.m – Computes the modified Newton-Raphson step for multi-dimensional parameter optimization, and computes the cost gradient and information matrix.

oe.m – Estimates dynamic model parameters from measured input-output time-domain data using output-error maximum likelihood. This routine uses general dynamic model definition m-files that can be linear or nonlinear.

senest.m – Computes sensitivity estimates using finite differences.

simplex.m – Implements the simplex method for multi-dimensional parameter optimization without cost function gradients.

Estimated Parameter Error Bounds

r_colores.m – Computes Cramér-Rao bounds for the covariances of the estimated parameters in a linear least squares regression formulation, both conventionally and accounting for colored residuals.

colores.m – Computes corrected parameter standard errors by post-processing results from output-error maximum likelihood parameter estimation (**oe.m**). The correction accounts for the practical fact that output-error residuals are usually colored, not white, as assumed in the output-error maximum likelihood theoretical formulation. Corrected parameter standard errors from **colores.m** are consistent with the scatter in parameter estimates from repeated flight test maneuvers, and therefore accurately represent estimated parameter uncertainty.

m_colores.m – Vectorized version of **colores.m**, which runs much faster and gives the same results as **colores.m**.

Real-Time Parameter Estimation

rft.m – Computes the recursive discrete Fourier transform of a time series.

rtpid.m – Computes sequential real-time estimates of dynamic model parameters and the associated covariance matrix, using equation-error in the frequency domain.

rlesq.m – Computes equation-error model parameter estimates and covariance matrix estimates for a linear least squares regression problem, using a recursive formulation. This routine is the recursive version of **lesq.m**.

slesq.m – Computes equation-error model parameter estimates and covariance matrix estimates for a linear least squares regression problem, using a batch formulation applied to sequential sections of data. This routine is the sequential version of **lesq.m**.

Frequency Domain Parameter Estimation

fdoe.m – Estimates dynamic model parameters from measured input-output frequency-domain data using output-error maximum likelihood. This routine is the frequency-domain equivalent of **oe.m**.

tfest.m – Estimates parameters in a transfer function model structure, using equation-error in the frequency domain and **fint.m** to compute high-accuracy finite Fourier integrals for arbitrary frequencies.

loest.m – Estimates parameters in a low order equivalent system transfer function model structure, using the same approach as in **tfest.m**, except that a relaxation technique is used to estimate the equivalent time delay parameter.

Utilities

ab3.m – Implements third-order Adams-Bashforth numerical integration.

adamb3.m – Version of **ab3.m** used in parameter estimation algorithms.

adamb3.c – C mex-file version of **adamb3.m** used for high speed computation in parameter estimation algorithms.

buzz.m – Adds white noise to a time series.

colnse.m – Adds a selectable combination of white noise and band-limited noise to a time series.

comfun.m – Computes the value of ordinary polynomial functions.

dband.m – Applies a dead band to a time series.

int1.m – Does one-dimensional linear interpolation.

int2.m – Does two-dimensional linear interpolation.

int3.m – Does three-dimensional linear interpolation.

loadflat.m – Reads a general ASCII flat file into the MATLAB® workspace.

massprop.m – Assembles aircraft mass and moment of inertia data.

misvd.m – Computes robust matrix inversion using singular value decomposition.

milstd.m – Computes longitudinal handling qualities level prediction according to MIL-STD 1797A, and plots handling qualities parameters and boundaries.

model_disp.m – Displays the functional form and parameter values for a polynomial model.

ocf.m – Converts a transfer function numerator and denominator into a state space model in observer canonical form.

plotmesh.m – Makes 3-D mesh plots for polynomial models.

plotpest.m – Plots parameter estimates and error bars representing 95 percent confidence intervals.

plotsurf.m – Makes 3-D surface plots for polynomial models.

plot3d.m – Makes 3-D plots using data arranged for linear regression.

press.m – Computes predicted sum of squares metric.

ratelim.m – Implements rate limits.

regcor.m – Computes and displays pairwise regressor correlations.

reggen.m – Generates multivariate polynomial regressors.

rms.m – Computes root mean square of the elements of a vector.

rk2.m – Implements second-order Runge-Kutta numerical integration.

runk2.m – Version of **rk2.m** used in parameter estimation algorithms.

runk2.c – C mex-file version of **runk2.m** used in parameter estimation algorithms.

rk4.m – Implements fourth-order Runge-Kutta numerical integration.

runk4.m – Version of **rk4.m** used in parameter estimation algorithms.

runk4.c – C mex-file version of **runk4.m** used in parameter estimation algorithms.

spl.m – Generates spline functions.

ulag.m – Applies a selected time shift to a time series.

Example

To demonstrate the use and capabilities of *SIDPAC*, a real flight test data analysis and modeling problem is presented. This example was chosen so that some of the main elements of *SIDPAC* could be demonstrated.

The flight test data was obtained from a longitudinal maneuver flown on the NASA Glenn Twin Otter aircraft, see Figure 1. The Twin Otter is a twin-engine turboprop commuter aircraft equipped with high-quality flight research instrumentation.

The flight test data for the example maneuver was available in an ASCII file, with the measured quantities named according to the convention used at NASA Glenn. To use *SIDPAC* effectively, the first step is to put the data in a standard format that some of the *SIDPAC* routines expect. This is a simple matter of unit conversions, routine calculations, and assembling the data into an array called **fdata** in the MATLAB® workspace, where each column of the array is assigned a specific quantity. There is a GUI in *SIDPAC* to make this process easier, which is invoked by simply typing **sid** at the MATLAB® command window prompt. The resulting GUI is shown in Figure 2. The analyst can use this GUI to quickly plot any of the workspace variables, convert units, then assign the result to the appropriate place in the standard data matrix **fdata**. A side benefit of this procedure is that the flight test data is automatically checked for missing information as the analyst assigns data to the channels in the standard data matrix **fdata**. In this example, the measurement for heading angle **psi** was missing. However, this measurement is not needed for longitudinal data analysis and modeling.

For large-scale flight test data analysis, the data reduction and channel assignment process is typically implemented by creating a script, so that the data reduction steps can be repeated automatically for each maneuver. A script is simply a text file containing commands that could have been issued at the MATLAB® command prompt. The commands in the script are executed by typing the name of the script file, omitting the file extension. Script files must have the **.m** file extension.

The analyst clicks on the **Next** button to proceed to the next GUI, shown in Figure 3, which can be used to cut the maneuver data length. This capability is necessary because it is common for the data to include more than is necessary for the analysis and dynamic modeling. For example, there may be long stretches of steady trim with no data information content, or it may be that the pilot applied a different power setting or changed the flight condition near the end of the data, to set up for the next flight test maneuver. In the latter case, the assumptions for the modeling may be violated, so this part of the data must be removed. The **Cut Maneuver** button allows the analyst to define the beginning and end of the maneuver using graphical (mouse) or numerical (keyboard) input.

Clicking the **Next** button again, the analyst arrives at the data compatibility analysis GUI, where the kinematic consistency of the measured output quantities is checked. Figure 4 shows data compatibility plots of the measured airspeed, angle of attack, and sideslip angle, compared to reconstructed values obtained from integrating the kinematic equations using translational accelerations and angular rate measurements as inputs. The analogous check on the Euler angle measurements, using the rotational kinematic equations with angular rate measurement inputs to reconstruct the Euler angles, is shown when the analyst selects the rotational option from the pull-down menu at the upper right of the GUI, see Figure 5.

SIDPAC software was designed so that it is possible to enter or exit any GUI at any time without adversely affecting results. The analyst can issue any required problem-specific commands in the MATLAB® command window (whether the current GUI is closed or not), then continue with the analysis using the GUI.

Instrumentation error parameters can be found using output-error parameter estimation and a default selection for the instrumentation error model structure by simply clicking the **Estimate Errors** button in the data compatibility GUI. Parameter estimation results are stored and displayed in the MATLAB® workspace. Estimated instrumentation errors can be removed from the measured data by clicking the **Correct Data** button. Figure 6 shows the rotational data after the estimated instrumentation corrections have been applied to the data. Figures 5 and 6 show how applying the estimated instrumentation error corrections implements kinematic consistency among the measured outputs related to the pitch rotation.

At this point, the data have been organized, plotted, checked for data compatibility, and corrected using estimated instrumentation errors. The next steps vary greatly depending on the goals of the investigation. For this example, a script was developed to implement and document the data analysis and modeling process. Figure 7 shows a listing of the script. Space does not permit showing all of the results generated from this script, but a short description will be given here.

The initial commands in the script plot the measured data and compute non-dimensional aerodynamic coefficients. Next, body-axis Z force and pitching moment coefficients are modeled in an equation-error formulation using least squares regression and stepwise regression. Corrected parameter error bounds are computed and displayed, along with the parameter estimates and confidence intervals. Modeling is then carried out in the frequency domain, using a transfer function model and equation-error parameter estimation. Following this, output-error parameter estimation in the time domain is used with a dynamic model file implementing the full nonlinear equations of motion and a linear aerodynamic model to estimate non-dimensional stability and control derivatives. A prediction case is included to demonstrate the validity of the identified models and show the linear and nonlinear simulation capabilities. Plots of the results appear in the MATLAB® figure window and various displays of results appear in the MATLAB® command window.

Figure 8 shows one of the plots, which is a comparison of the measured non-dimensional pitching moment coefficient with the identified equation-error model. Table 1 contains parameter estimation results for this case.

The entire demonstration script in Figure 7 is available in *SIDPAC* as totter_demo.m. The demonstration can be executed by simply typing totter_demo at the MATLAB® command prompt.

Concluding Remarks

A collection of computer programs called System IDentification Programs for AirCraft, or *SIDPAC*, was described and demonstrated on a real flight test data analysis and modeling problem. *SIDPAC* was developed and tested at NASA Langley Research Center in the course of solving real problems in aircraft system identification.

SIDPAC addresses a wide range of system identification problems in a common MATLAB® environment. MATLAB® has many advantages, including platform-independence, easy to learn and program, many built-in functions for linear algebra, data analysis, debugging, plotting, and data visualization, and the use of double precision arithmetic by default. These characteristics make MATLAB® an excellent choice for implementing *SIDPAC* algorithms.

Relevant theory and practical considerations for the methods implemented in *SIDPAC* are completely described in the forthcoming text entitled *System Identification Applied to Aircraft – Theory and Practice*, by V. Klein and E.A. Morelli. The *SIDPAC* software is a product of NASA Langley Research Center, and is available free of charge to U.S. citizens by request to the author, contingent on the requestor completing a NASA software usage agreement. Requests for the software should be e-mailed to:

e.a.morelli@larc.nasa.gov

Please include name, mailing address, e-mail address, telephone number, and company affiliation of the requestor, along with a one-sentence description of the intended use of *SIDPAC*, for technology transfer record-keeping purposes.

SIDPAC allows a user to apply state-of-the-art technology to aircraft system identification problems, within a single, highly capable, and easy-to-use computing environment.

References

Data Analysis

Bendat, J.S. and Piersol, A.G. *Random Data Analysis and Measurement Procedures,* 2nd Ed., John Wiley & Sons, New York, NY, 1986.

Gainer, T.G. and Hoffman, S. "Summary of Transformation Equations and Equations of Motion Used in Free-Flight and Wind-Tunnel Data Reduction Analysis," NASA SP-3070, 1972.

Graham, R.J., "Determination and Analysis of Numerical Smoothing Weights," NASA TR R-179, December 1963.

Lanczos, C. *Applied Analysis*, Dover Publications, Inc., New York, NY, 1988.

Morelli, E.A. "Estimating Noise Characteristics from Flight Test Data using Optimal Fourier Smoothing," *Journal of Aircraft*, Vol. 32, No. 4, July-August 1995, pp. 689-695.

Data Compatibility Analysis

Klein, V. and Morgan, D.R. "Estimation of Bias Errors in Measured Airplane Responses using Maximum Likelihood Method," NASA TM 89059, January 1987.

Morelli, E. A., "Optimal Input Design for Aircraft Instrumentation Systematic Error Estimation," AIAA paper 91-2850, *AIAA Atmospheric Flight Mechanics Conference*, New Orleans, Louisiana, August 1991.

Model Structure Determination

Draper, N.R. and Smith, H. *Applied Regression Analysis*, 2nd Edition, John Wiley & Sons, New York, NY, 1981.

Klein, V., Batterson, J.G., and Murphy, P.C. "Determination of Airplane Model Structure from Flight Data by using Modified Stepwise Regression," NASA TP-1916, October 1981.

Morelli, E.A., "Global Nonlinear Aerodynamic Modeling using Multivariate Orthogonal Functions," *Journal of Aircraft*, Vol. 32, No. 2, March-April 1995, pp. 270-77.

Morelli, E.A., "Global Nonlinear Parametric Modeling with Application to F-16 Aerodynamics," ACC paper WP04-2, Paper ID i-98010-2, *American Control Conference*, Philadelphia, Pennsylvania, June 1998.

Morelli, E.A. and DeLoach, R., "Response Surface Modeling using Multivariate Orthogonal Functions," AIAA paper 2001-0168, *39th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, January 2001.

Estimated Parameter Error Bounds

Maine, R.E. and Iliff, K.W., "The Theory and Practice of Estimating the Accuracy of Dynamic Flight-Determined Coefficients," NASA RP 1077, July 1981.

Morelli, E.A. and Klein, V. "Accuracy of Aerodynamic Model Parameters Estimated from Flight Test Data," *Journal of Guidance, Control, and Dynamics*, Vol. 20, No. 1, January-February 1997, pp. 74-80.

Equation-Error Parameter Estimation

Klein, V., "Estimation of Aircraft Aerodynamic Parameters from Flight Data," *Prog. Aerospace Sciences*, Vol. 26, 1989, pp. 1-77.

Klein, V. and Morelli, E. A., "Parameter Estimation of a Highly Augmented Aircraft from Flight Data," *9th IFAC/IFORS Symposium on Identification and System Parameter Estimation*, Budapest, Hungary, July 1991.

Output-Error Parameter Estimation

Maine, R.E. and Iliff, K.W., "Application of Parameter Estimation to Aircraft Stability and Control - The Output-Error Approach", NASA RP 1168, June 1986.

Taylor, L.W., Jr. and Iliff, K.W. "Systems Identification using a Modified Newton-Raphson Method – A Fortran Program," NASA TN D-6734, May 1972.

Morelli, E.A. and Klein, V. "Determining the Accuracy of Maximum Likelihood Parameter Estimates with Colored Residuals," NASA CR 194893, March 1994.

Low Order Equivalent System Identification

Military Standard – Flying Qualities of Piloted Aircraft, MIL-STD-1797A, January 1990.

Morelli, E.A., "Low Order Equivalent System Identification for the Tu-144LL Supersonic Transport Aircraft," AIAA paper 2000-3902, *AIAA Atmospheric Flight Mechanics Conference*, Denver, Colorado, August 2000.

Morelli, E.A. "Identification of Low Order Equivalent System Models from Flight Test Data," NASA TM-2000-210117, August 2000.

Parameter Estimation in the Frequency Domain

Klein, V. "Aircraft Parameter Estimation in Frequency Domain," AIAA paper 78-1344, *AIAA Atmospheric Flight Mechanics Conference*, Palo Alto, CA, August 1978.

Morelli, E.A. "High Accuracy Evaluation of the Finite Fourier Transform using Sampled Data," NASA TM 110340, June 1997.

Real-Time Parameter Estimation

Morelli, E.A. "In-Flight System Identification," AIAA paper 98-4261, *AIAA Atmospheric Flight Mechanics Conference*, Boston, MA, August 1998.

Morelli, E.A., "Real-Time Parameter Estimation in the Frequency Domain," *Journal of Guidance, Control, and Dynamics*, Vol. 23, No. 5, September-October 2000, pp. 812-818.

Experiment Design

Box, G.E.P., Hunter, W.G., and Hunter, J.S., *Statistics for Experimenters – An Introduction to Design, Data Analysis, and Model Building*, John Wiley & Sons, Inc., New York, NY, 1978.

Morelli, E.A., "Practical Input Optimization for Aircraft Parameter Estimation Experiments," NASA CR 191462, May 1993.

Morelli, E.A., "Flight Test of Optimal Inputs and Comparison with Conventional Inputs," *Journal of Aircraft*, Vol. 36, No. 2, March-April 1999, pp. 389-397.

Schroeder, M.R. "Synthesis of Low-Peak-Factor Signals and Binary Sequences with Low Autocorrelation," *IEEE Transactions on Information Theory*, January 1970, pp. 85-89.

Williams, J.N., Ham, J.A., and Tischler, M.B., "Flight Test Manual, Rotorcraft Frequency Domain Flight Testing," U.S. Army Aviation Technical Test Center, Edwards AFB, CA, AQTD Project No. 93-14, September 1995.

Software Implementation

Using MATLAB®, Version 6, The MathWorks, Inc., Natick, MA, 2000.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. *Numerical Recipes – The Art of Scientific Computing (FORTRAN Version)*, Cambridge University Press, Cambridge, UK, 1989.

Table 1 Twin Otter Equation-Error Modeling Results

Parameter	Estimate	Std. Error
C_{m_α} (rad^{-1})	−1.476	0.022
C_{m_q}	−36.35	0.69
$C_{m_{\delta_e}}$ (rad^{-1})	−1.869	0.020
C_{m_o}	0.0023	0.0001

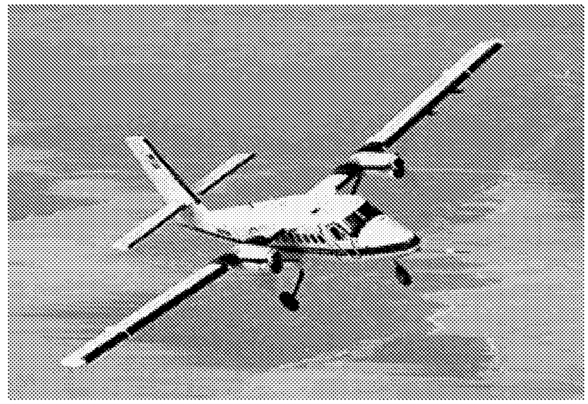

Figure 1 NASA Glenn Twin Otter

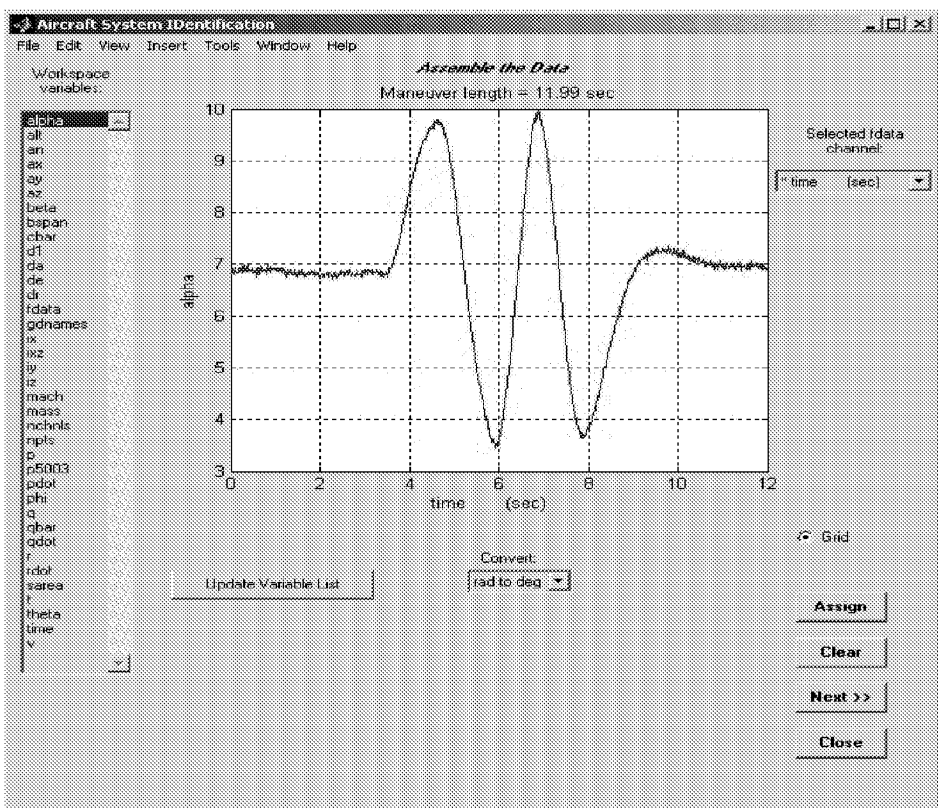

Figure 2 *SIDPAC* Data Channel Assignment GUI

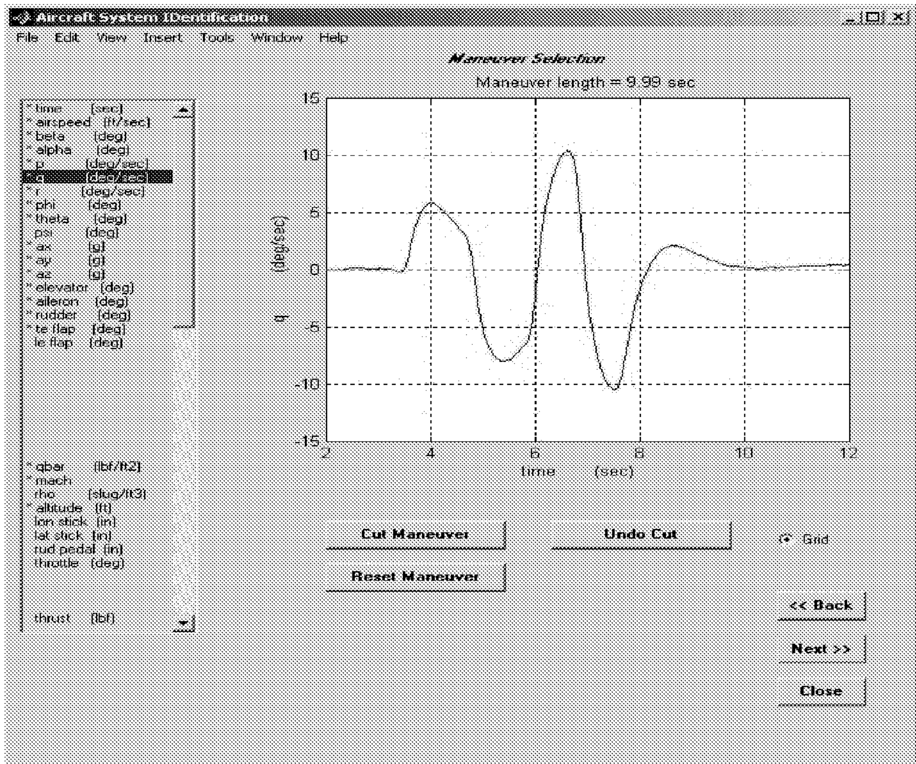

Figure 3 *SIDPAC* Maneuver Length GUI

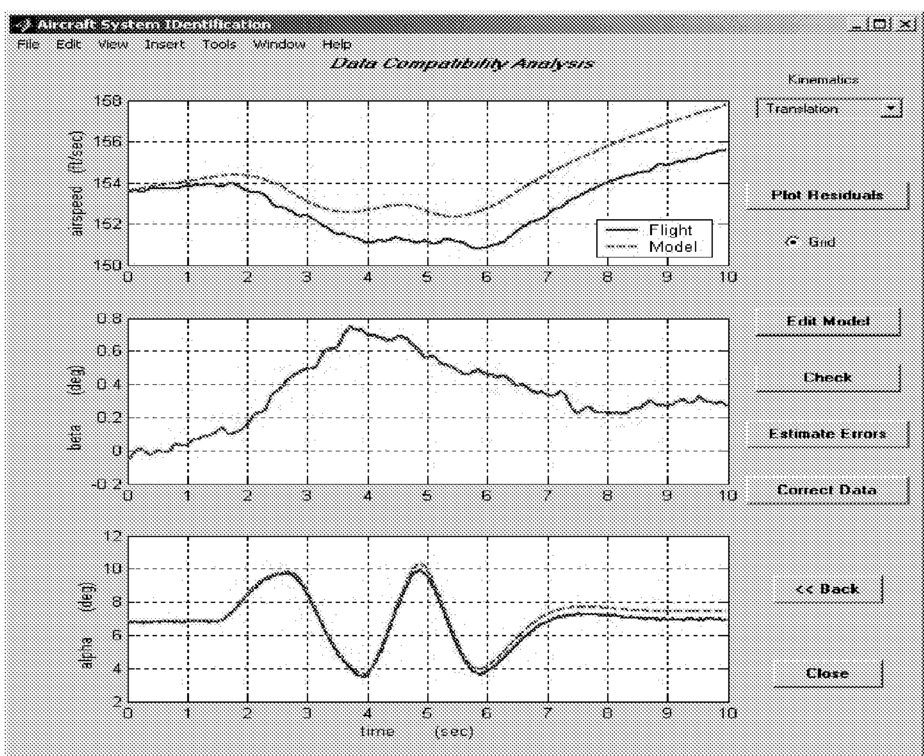

Figure 4 *SIDPAC* Translational Data Compatibility GUI

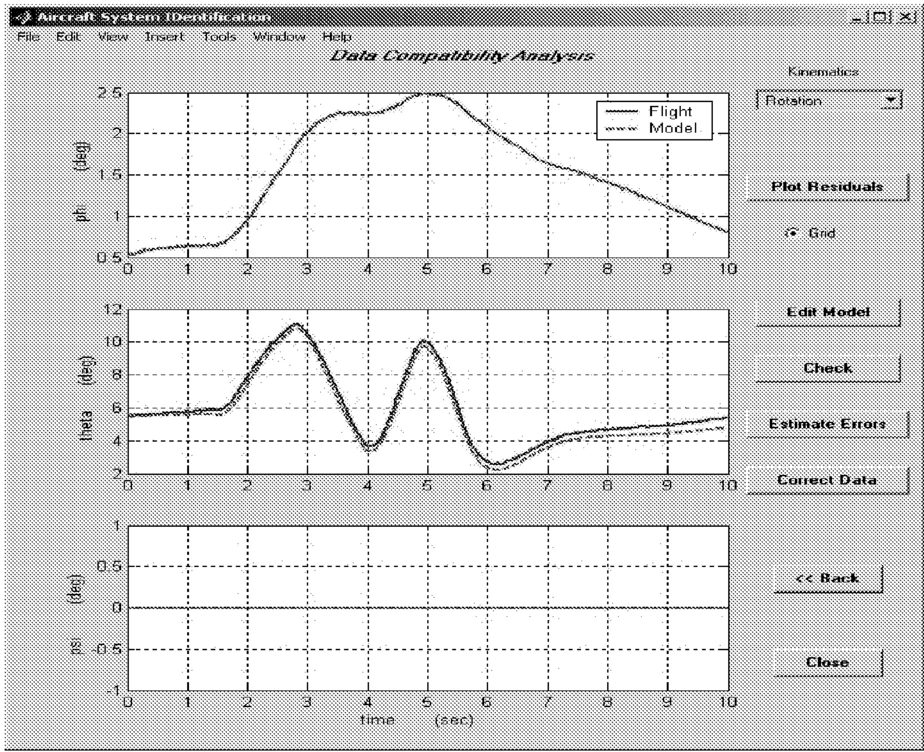

Figure 5 *SIDPAC* Rotational Data Compatibility GUI

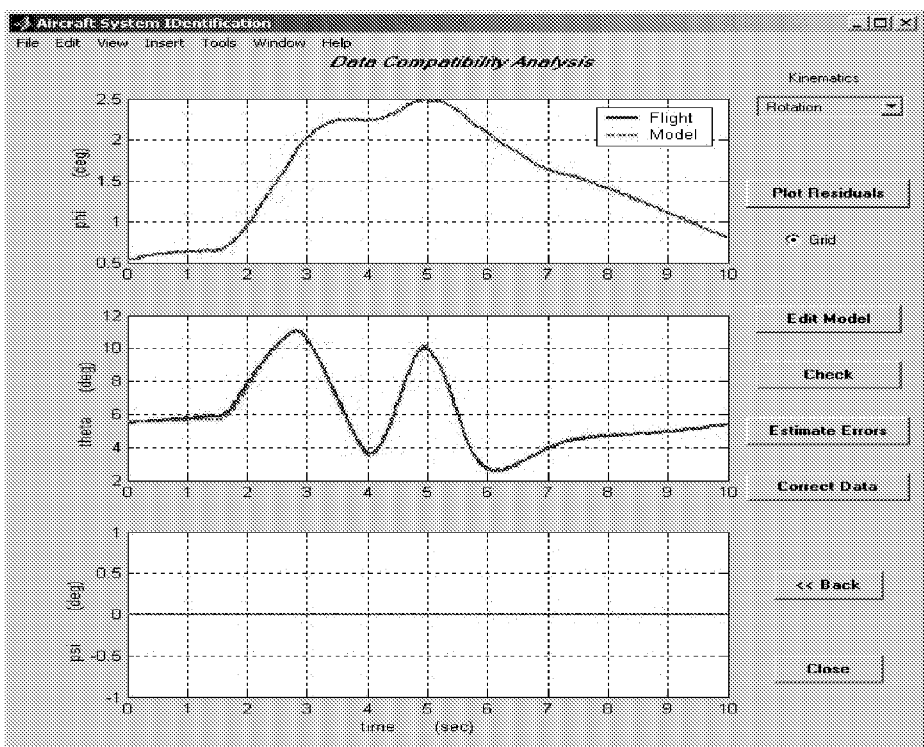

Figure 6 *SIDPAC* Rotational Data Compatibility GUI with Instrumentation Corrections Applied

```
%
% script totter_demo.m
%
% Usage: totter_demo;
%
% Description:
%
%   Demonstrates flight data analysis and modeling
%   using SIDPAC for a longitudinal flight test maneuver
%   on the NASA Glenn Twin Otter aircraft.
%
% Input:
%
%   None
%
% Output:
%
%   data file
%   2-D plots
%

%
%   Calls:
%     compfc.m
%     compmc.m
%     xsmep.m
%     lesq.m
%     r_colores.m
%     model_disp.m
%     swr.m
%     zep.m
%     tfest.m
%     nldyn_psel.m
%     oe.m
%     nldyn.m
%     m_colores.m
%     plotpest.m
%     tfsim.m
%
%   Author:  Eugene A. Morelli
%
%   History:
%    11 Jul 2002 - Created and debugged, EAM.
%
% Copyright (C) 2002  Eugene A. Morelli
%
% This program carries no warranty, not even the implied
% warranty of merchantability or fitness for a particular purpose.
%
% Please email bug reports or suggestions for improvements to:
%
%     e.a.morelli@larc.nasa.gov
%

%
% Load the data file.
%
load 'totter_lon_020213f1_018.mat'
%
% Set up the figure window.
%
FgH=figure('Units','normalized','Position',[.506 .231 .504 .715],...
          'Name','SIDPAC Demonstration','NumberTitle','off','Toolbar','none');
%
% Plot the measured inputs and outputs.
%
subplot(4,1,1), plot(t,fdata(:,14),'LineWidth',2),
title('Twin Otter Flight Test Data','FontWeight','bold'),
grid on, ylabel('elevator  (deg)'),
```

Figure 7 Twin Otter Data Analysis and Modeling Script (continued)

```
subplot(4,1,2), plot(t,fdata(:,4),'LineWidth',2),
grid on, ylabel('alpha  (deg)'),
subplot(4,1,3), plot(t,fdata(:,6),'LineWidth',2),
grid on, ylabel('q  (deg/sec)'),
subplot(4,1,4), plot(t,fdata(:,13),'LineWidth',2),
grid on, ylabel('az  (g)'), xlabel('Time (sec)'),
fprintf('\n\n The figure shows the measured input and outputs.')
fprintf('\n\n Press any key to continue ... '),pause,
%
%  Calculate aerodynamic force and moment coefficients.
%
fprintf('\n\n Calculate the non-dimensional ')
fprintf('\n aerodynamic force and moment ')
fprintf('\n coefficients using compfc.m and compmc.m:')
fprintf('\n\n [CX,CY,CZ,CD,CYw,CL]=compfc(fdata);')
fprintf('\n\n [Cl,Cm,Cn]=compmc(fdata);')
[CX,CY,CZ,CD,CYw,CL]=compfc(fdata);
[Cl,Cm,Cn,pv,qv,rv]=compmc(fdata);
subplot(2,1,1),plot(t,CZ,'LineWidth',2),grid on,ylabel('Z Force Coefficient'),
title('Non-Dimensional Coefficients from Flight Test Data','FontWeight','bold'),
subplot(2,1,2),plot(t,Cm,'LineWidth',2),grid on,ylabel('Pitching Moment Coefficient'),xlabel('Time (sec)'),
fprintf('\n\n Press any key to continue ... '),pause,
%
%  Assemble the regressor matrix.
%
fprintf('\n\n Assemble the matrix of regressors ')
fprintf('\n for equation-error parameter estimation: ')
fprintf('\n\n alpha  (rad)'),
fprintf('\n qhat '),
fprintf('\n elevator  (rad)'),
X=[fdata(:,4)*pi/180,fdata(:,72),fdata(:,14)*pi/180];
%
%  Plot the regressors.
%
subplot(3,1,1),plot(t,X(:,1),'LineWidth',2),grid on,ylabel('alpha  (rad)'),
title('Equation-Error Regressors','FontWeight','bold'),
subplot(3,1,2),plot(t,X(:,2),'LineWidth',2),grid on,ylabel('qhat '),
subplot(3,1,3),plot(t,X(:,3),'LineWidth',2),grid on,ylabel('elevator  (rad)'),
xlabel('Time (sec)'),
fprintf('\n\n Press any key to continue ... '),pause,
%
%  Find smoothed trim values.
%
fprintf('\n\n Find the smoothed trim values ')
fprintf('\n from the regressors using xsmep.m:')
fprintf('\n\n X0=xsmep(X,1.0,dt);')
X0=xsmep(X,1,dt);
%
%  Plot the regressors and the smoothed trim values.
%
subplot(3,1,1),plot(t,X(:,1),'LineWidth',2),hold on,
title('Equation-Error Regressors','FontWeight','bold'),
plot(t(1),X(1,1),'r.','MarkerSize',14,'LineWidth',2), hold off,
grid on,ylabel('alpha  (rad)'),
subplot(3,1,2),plot(t,X(:,2),'LineWidth',2), hold on,
plot(t(1),X(1,2),'r.','MarkerSize',14,'LineWidth',2), hold off,
grid on,ylabel('qhat '),
subplot(3,1,3),plot(t,X(:,3),'LineWidth',2), hold on,
plot(t(1),X(1,3),'r.','MarkerSize',14,'LineWidth',2), hold off,
grid on,ylabel('elevator  (deg)'),xlabel('Time (sec)'),
%
%  Remove the smoothed trim values.
%
fprintf('\n\n Remove the smoothed trim values ')
fprintf('\n from the regressors using :')
fprintf('\n\n X=X-ones(size(X,1),1)*X0(1,:);')
X=X-ones(size(X,1),1)*X0(1,:);
%
%  Program lesq.m requires a constant regressor for the bias term.
%
```

Figure 7 Twin Otter Data Analysis and Modeling Script (continued)

```
X=[X,ones(size(X,1),1)];
fprintf('\n\n Press any key to continue ... '),pause,
%
% Linear regression for the Z force coefficient.
%
fprintf('\n\n Z force coefficient:')
fprintf('\n\n Estimate stability and control ')
fprintf('\n derivatives using equation-error ')
fprintf('\n linear regression program lesq.m: ')
fprintf('\n\n [yZ,pZ,crbZ,s2Z]=lesq(X,CZ);')
[yZ,pZ,crbZ,s2Z]=lesq(X,CZ);
%
% Plot the results.
%
subplot(2,1,1),plot(t,CZ,t,yZ,'r:','LineWidth',2),grid on,
title('Equation-Error Parameter Estimation','FontWeight','bold'),
ylabel('CZ'),legend('Flight data','Regression model',0),
subplot(2,1,2),plot(t,CZ-yZ,'LineWidth',1.5),grid on,
ylabel('Residual'),xlabel('Time (sec)'),
%
% Compute and display the error bounds.
%
fprintf('\n\n Compute the estimated parameter ')
fprintf('\n error bounds using r_colores.m: ')
fprintf('\n\n [crbZ,crboZ]=r_colores(X,CZ); ')
[crbZ,crboZ]=r_colores(X,CZ);
serroZ=sqrt(diag(crboZ));
serrZ=sqrt(diag(crbZ));
perrZ=100*serrZ./abs(pZ);
fprintf('\n\n Display the parameter estimation ')
fprintf('\n results using model_disp.m:')
Xlab=['alpha  (rad)   ';'qhat           ';'elevator  (rad)'];
model_disp(pZ,serrZ,[1,10,100,0],Xlab);
fprintf('\n\n Press any key to continue ... '),pause,
%
% Stepwise regression for the pitching moment coefficient.
%
fprintf('\n\n Pitching moment coefficient: ')
fprintf('\n\n Add a nonlinear cross term alpha*elevator ,')
fprintf('\n regressor and use stepwise regression program swr.m:')
fprintf('\n\n [ym,pm,crbm,s2m]=swr(X,Cm);')
%
% Program swr.m adds the bias term automatically,
% so the constant regressor is not necessary.  Add
% the nonlinear cross term to the regressor matrix X.
%
X=[X(:,[1:3]),X(:,1).*X(:,3)];
[ym,pm,crbm,s2m,Xm,pindxm]=swr(X,Cm,1);
%
% Include only parameters for selected regressors.
%
pm=pm(pindxm);
%
% Plot the results.
%
subplot(2,1,1),plot(t,Cm,t,ym,'r:','LineWidth',1.5),grid on,
title('Pitching Moment Coefficient','FontWeight','bold'),
ylabel('Cm'),legend('Flight data','Equation-Error model')
subplot(2,1,2),plot(t,Cm-ym,'LineWidth',1.5),grid on,
ylabel('Residual'),xlabel('Time (sec)'),
%
% Compute and display the error bounds.
%
fprintf('\n\n Compute the estimated parameter ')
fprintf('\n error bounds using r_colores.m: ')
fprintf('\n\n [crbm,crbom]=r_colores(X,Cm); ')
[crbm,crbom]=r_colores(Xm,Cm);
serrom=sqrt(diag(crbom));
serrm=sqrt(diag(crbm));
perrm=100*serrm./abs(pm);
```

Figure 7 Twin Otter Data Analysis and Modeling Script (continued)

```
fprintf('\n\n Display the parameter estimation ')
fprintf('\n results using model_disp.m:')
model_disp(pm,serrm,[1,10,100,0],Xlab);
fprintf('\n\n Press any key to continue ... '),pause,
%
% Estimate the transfer function model q/de.
%
fprintf('\n\n Estimate the transfer function ')
fprintf('\n for pitch rate to elevator deflection ')
fprintf('\n (q/de), using tfest.m:')
fprintf('\n\n [ytf,num,den,ptf,crbtf,s2tf,zr,xr,f] = tfest(u,z,t,1,2,w);')
fprintf('\n\n The frequency vector is w = 2*pi*[0.3:.01:1.3]''.')
w=2*pi*[0.3:.01:1.3]';
%
% Detrend the time domain data for frequency domain analysis.
%
u=zep(fdata(:,14));
z=zep(fdata(:,6));
subplot(2,1,1),plot(t,u,'LineWidth',2),grid on,
title('Transfer Function Modeling Data','FontWeight','bold'),
ylabel('Elevator  (deg)'),
subplot(2,1,2),plot(t,z,'LineWidth',2),grid on,
ylabel('Pitch Rate  (deg/sec)'),xlabel('Time (sec)'),
fprintf('\n\n Press any key to continue ... '),pause,
[ytf,num,den,ptf,crbtf,s2tf,zr,xr,f] = tfest(u,z,t,1,2,w);
subplot(2,1,1),plot(f,abs(zr),f,abs(xr*ptf),'r:','LineWidth',1.5),grid on,
title('Frequency Domain Transfer Function Modeling','FontWeight','bold'),
ylabel('Magnitude'),legend('Flight data','Transfer function model')
subplot(2,1,2),plot(f,unwrap(angle(zr)),f,unwrap(angle(xr*ptf)),'r:','LineWidth',1.5),grid on,
ylabel('Phase'),xlabel('Frequency (Hz)'),
fprintf('\n'),tf(num,den),
fprintf('\n\n The figure shows the frequency domain fit. ')
fprintf('\n\n Identified modes from the transfer function ')
fprintf('\n identification in the frequency domain are: \n')
damp(den),
fprintf('\n\n Press any key to continue ... '),pause,
subplot(2,1,1),plot(t,z,t,ytf,'r:','LineWidth',1.5),grid on,
title('Equation-Error Frequency Domain Transfer Function Modeling','FontWeight','bold'),
ylabel('Pitch Rate (deg/sec)'),legend('Flight data','Transfer function model')
subplot(2,1,2),plot(t,z-ytf,'LineWidth',2),grid on,
ylabel('Residual'),xlabel('Time (sec)'),
fprintf('\n\n The figure now shows the time domain fit. ')
fprintf('\n\n Press any key to continue ... '),pause,
%
% Estimate the dimensional stability and control derivatives
% using time-domain output-error parameter estimation.
%
fprintf('\n\n\n Now estimate the non-dimensional stability ')
fprintf('\n and control derivatives using output-error ')
fprintf('\n parameter estimation in the time domain.')
fprintf('\n\n Input:   elevator (rad)')
fprintf('\n Outputs: alpha (rad), q (rad/sec), az (g)')
dtr=pi/180;
u=fdata(:,[14:16])*dtr;
z=[fdata(:,[4,6])*dtr,fdata(:,13)];
%
% Plot the measured inputs and outputs.
%
subplot(4,1,1), plot(t,u(:,1),'LineWidth',2),
title('Output-Error Time Domain Modeling','FontWeight','bold'),
grid on, ylabel('elevator  (rad)'),
subplot(4,1,2), plot(t,z(:,1),'LineWidth',2),
grid on, ylabel('alpha  (rad)'),
subplot(4,1,3), plot(t,z(:,2),'LineWidth',2),
grid on, ylabel('q  (rad/sec)'),
subplot(4,1,4), plot(t,z(:,3),'LineWidth',2),
grid on, ylabel('az  (g)'), xlabel('Time (sec)'),
fprintf('\n\n The figure shows the measured input and outputs.')
```

Figure 7 Twin Otter Data Analysis and Modeling Script (continued)

```
%
% Set up for the output-error parameter estimation using
% nldyn.m to define the dynamic model.
%
nldyn_psel;
fprintf('\n\n Press any key to continue ... '),pause,
%
% Find initial parameter values for the
% output-error parameter estimation.
%
fprintf('\n\n Initial values of the parameters in ')
fprintf('\n vector p0 are obtained from the ')
fprintf('\n equation-error solution:\n')
%
% Omit the CZq term in the output-error formulation,
% because of low sensitivity at low angles of attack.
%
p0=[pZ([1,3,4]);pm],
serr0=[serrZ([1,3,4]);serrm];
fprintf('\n\n Estimate the model parameters ')
fprintf('\n using output-error parameter estimation ')
fprintf('\n program oe.m and dynamic model file nldyn.m:  ')
fprintf('\n\n [y,p,crb,rr]=oe(''nldyn'',p0,u,t,x0,cc,z);')
fprintf('\n\n Press any key to continue ... '),pause,
fprintf('\n\n Starting oe.m ...')
tic,[y,p,crb,rr]=oe('nldyn',p0,u,t,x0,cc,z);toc,
%
% Plot the results.
%
clf, title('Output-Error Parameter Estimation','FontWeight','bold'),
subplot(3,1,1),plot(t,z(:,1),t,y(:,1),'r:','LineWidth',2),grid on,ylabel('alpha  (rad)'),
legend('Flight data','Output-Error model',0),
subplot(3,1,2),plot(t,z(:,2),t,y(:,2),'r:','LineWidth',2),grid on,ylabel('q  (rad/sec)'),
subplot(3,1,3),plot(t,z(:,3),t,y(:,3),'r:','LineWidth',2),grid on,ylabel('az  (g)'),xlabel('Time (sec)'),
fprintf('\n The plots show the measured output data ')
fprintf('\n and the identified model fit. ')
fprintf('\n\n Press any key to continue ... '),pause,
%
% Examine the residuals.
%
clf, subplot(3,1,1),plot(t,z(:,1)-y(:,1),'LineWidth',2),grid on;ylabel('alpha residuals (rad)'),
title('Residuals','FontSize',12,'FontWeight','bold'),
subplot(3,1,2),plot(t,z(:,2)-y(:,2),'LineWidth',2),grid on;ylabel('q residuals (rad/sec)'),
subplot(3,1,3),plot(t,z(:,3)-y(:,3),'LineWidth',2),grid on;ylabel('az residuals (g)'),xlabel('Time (sec)'),
%
% Correct the estimated parameter error bounds.
%
fprintf('\n\n The output residuals are colored ')
fprintf('\n (due to modeling error), so the ')
fprintf('\n Cramer-Rao bounds calculated by oe.m must ')
fprintf('\n be corrected for colored residuals using ')
fprintf('\n program m_colores.m:')
fprintf('\n\n [crb,crbo] = m_colores(''nldyn'',p,u,t,x0,c,z);')
fprintf('\n\n Press any key to continue ... '),pause,
fprintf('\n\n Starting m_colores.m ...\n\n')
tic,[crb,crbo] = m_colores('nldyn',p,u,t,x0,cc,z);toc,
serr=sqrt(diag(crb));
%
% Display the parameter estimation results.
%
model_disp(p,serr,[1,100,0,1,10,100,0],Xlab);
leglab=['Equation-Error';'Output-Error  '];
parlab=['CZ_alpha';'CZ_de   ';'CZ_o    ';...
      'Cm_alpha';'Cm_q    ';'Cm_de   ';'Cm_o    '];
indx=[4,6]';
plotpest([p0(indx),p(indx)],[serr0(indx),serr(indx)],[],[],parlab(indx,:),leglab);
title('Parameter Estimation Results','FontWeight','bold')
fprintf('\n\n The figure shows that the equation-error ')
fprintf('\n and output-error parameter estimates are ')
fprintf('\n in statistical agreement. ')
```

Figure 7 Twin Otter Data Analysis and Modeling Script (continued)

```
fprintf('\n\n Press any key to continue ... '),pause,
save 'totter_results.mat' num den p serr p0 serr0 pZ serrZ pm serrm cc dtr;
%
% Check the prediction capability.
%
load 'totter_lon_020213f1_017.mat'
fprintf('\n\n Now check the prediction capability ')
fprintf('\n using data from a different maneuver ')
fprintf('\n and the identified transfer function ')
fprintf('\n model from before:')
fprintf('\n'),tf(num,den),
u=fdata(:,14); u=zep(u);
z=fdata(:,6); z=zep(z);
ytfp=tfsim(num,den,0,u,t);
%
% Plot the transfer function prediction results.
%
subplot(2,1,1),plot(t,z,t,ytfp,'r:','LineWidth',2),grid on,
title('Transfer Function Prediction','FontWeight','bold'),
ylabel('Pitch Rate (deg/sec)'),legend('Flight data','Transfer function prediction',4)
subplot(2,1,2),plot(t,z-ytfp,'LineWidth',2),grid on,
ylabel('Residual'),xlabel('Time (sec)'),
fprintf('\n\n The figure shows the time domain prediction ')
fprintf('\n using the transfer function model identified ')
fprintf('\n using data from a different maneuver. ')
fprintf('\n\n Press any key to continue ... '),pause,
u=fdata(:,[14:16])*dtr;
z=[fdata(:,[4,6])*dtr,fdata(:,13)];
nldyn_psel;
yp=nldyn(p,u,t,x0,cc);
%
% Plot the measured inputs and outputs.
%
subplot(4,1,1), plot(t,u(:,1),'LineWidth',2),
title('Twin Otter Flight Test Data','FontWeight','bold'),
grid on, ylabel('elevator  (rad)'),
subplot(4,1,2), plot(t,z(:,1),'LineWidth',2),
grid on, ylabel('alpha  (rad)'),
subplot(4,1,3), plot(t,z(:,2),'LineWidth',2),
grid on, ylabel('q  (rad/sec)'),
subplot(4,1,4), plot(t,z(:,3),'LineWidth',2),
grid on, ylabel('az  (g)'), xlabel('Time (sec)'),
fprintf('\n\n\n The figure shows the measured input and outputs ')
fprintf('\n for the prediction maneuver. ')
fprintf('\n\n Press any key to continue ... '),pause,
%
% Plot the output-error prediction results.
%
title('Output-Error Prediction','FontWeight','bold'),
%
% Correct for measurement biases.
%
bias=ones(length(t),1)\(z-yp);
yp=yp+ones(length(t),1)*bias;
subplot(3,1,1),plot(t,z(:,1)/dtr,t,yp(:,1)/dtr,'r:','LineWidth',2),grid on,ylabel('alpha  (rad)'),
legend('Flight data','Output-Error prediction',4),
title('Output-Error Model Prediction','FontWeight','bold'),
subplot(3,1,2),plot(t,z(:,2)/dtr,t,yp(:,2)/dtr,'r:','LineWidth',2),grid on,ylabel('q  (rad/sec)'),
subplot(3,1,3),plot(t,z(:,3),t,yp(:,3),'r:','LineWidth',2),grid on,ylabel('az  (g)'),xlabel('Time (sec)'),
fprintf('\n The plots show the measured output data ')
fprintf('\n and the prediction using the output-error ')
fprintf('\n model identified using data from a different ')
fprintf('\n maneuver. ')
fprintf('\n\n\nEnd of demonstration \n\n')
return
```

Figure 7 Twin Otter Data Analysis and Modeling Script (complete)

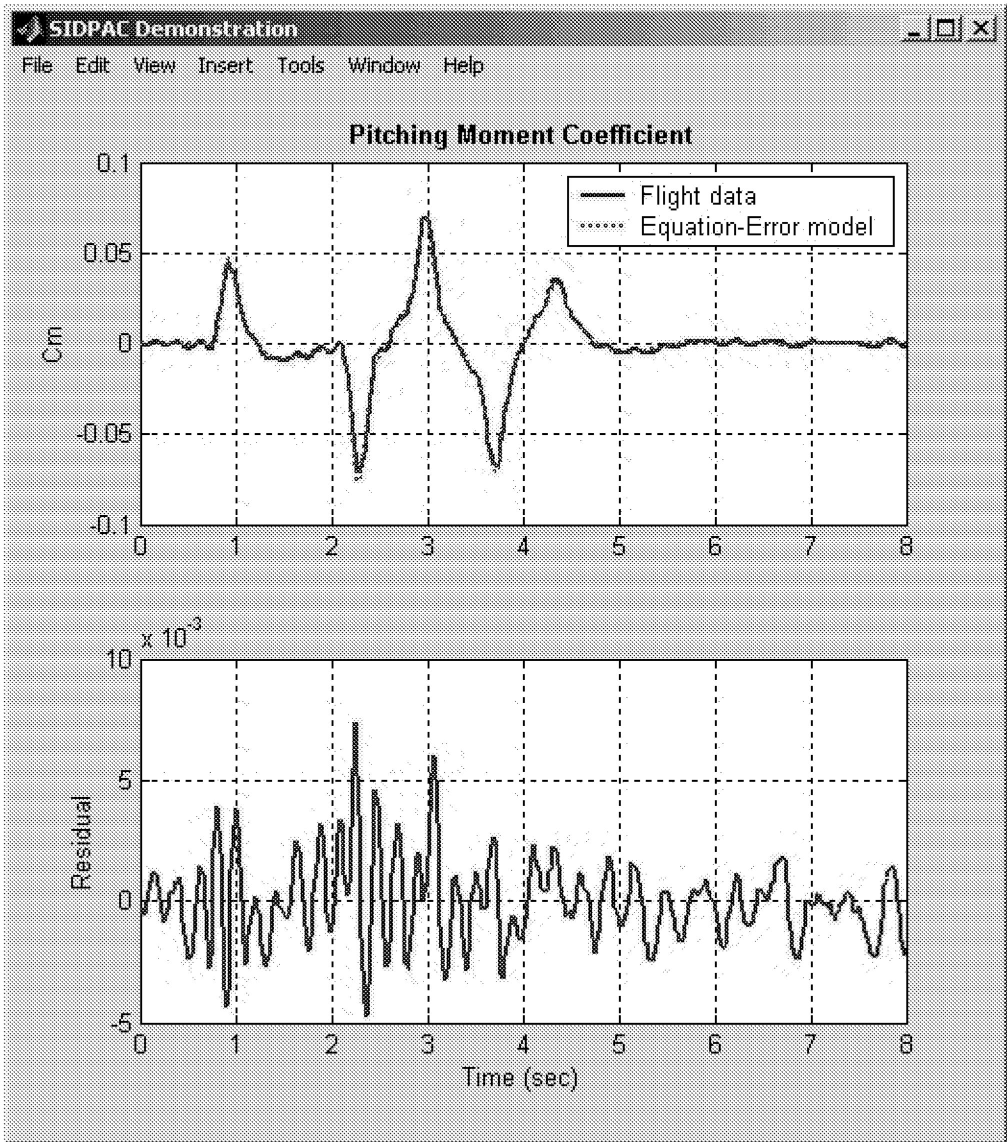

Figure 8 Twin Otter Pitching Moment Coefficient Modeling