User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)

Article · January 2007

CITATIONS
268

READS
4,889

3 authors, including:

Jonathan DeCastro
Toyota Research Institute
57 PUBLICATIONS **1,057** CITATIONS
SEE PROFILE

Jonathan Litt
NASA
97 PUBLICATIONS **1,720** CITATIONS
SEE PROFILE

NASA/TM—2007-215026

User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)

Dean K. Frederick
Saratoga Control Systems, Inc., Saratoga Springs, New York

Jonathan A. DeCastro
ASRC Aerospace Corporation, Cleveland, Ohio

Jonathan S. Litt
U.S. Army Research Laboratory, Glenn Research Center, Cleveland, Ohio

October 2007

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question via the Internet to *help@sti.nasa.gov*

- Fax your question to the NASA STI Help Desk at 301–621–0134

- Telephone the NASA STI Help Desk at 301–621–0390

- Write to:
  NASA Center for AeroSpace Information (CASI)
  7115 Standard Drive
  Hanover, MD 21076–1320

NASA/TM—2007-215026

User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)

Dean K. Frederick
Saratoga Control Systems, Inc., Saratoga Springs, New York

Jonathan A. DeCastro
ASRC Aerospace Corporation, Cleveland, Ohio

Jonathan S. Litt
U.S. Army Research Laboratory, Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

October 2007

Acknowledgments

The authors appreciate the assistance of the following past and present Glenn Research Center personnel: Donald Simon, Tak Kobayashi, Tom Lavelle, Scott Jones, Chris Snyder, Ten-Huei Guo, Sanjay Garg, Khary Parker, and Javad Sanati. The Matlab code used for the fan-speed controller designs is a version of the code in the GE ISICLE package that had been modified by the author in the course of work done for GE. Permission to use this code for this design GUI has been granted by GE, and is gratefully acknowledged. Shreeder Adibhatla of GE has been especially helpful in a number of ways. His suggestions as to the use of percent corrected fan speed for power management and for gains scheduling were particularly helpful. N&R Engineering and Management Services Corporation of Parma Heights, Ohio has acted as the main contractor for this work. The assistance of its president, Vinod Nagpal, is gratefully acknowledged.

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA Center for Aerospace Information
7115 Standard Drive
Hanover, MD 21076–1320

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161

Available electronically at http://gltrs.grc.nasa.gov

Contents

User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)

Dean K. Frederick
Saratoga Control Systems, Inc.
Saratoga Springs, New York 12866

Jonathan A. DeCastro
ASRC Aerospace Corporation
Cleveland, Ohio 44135

Jonathan S. Litt
U.S. Army Research Laboratory
Glenn Research Center
Cleveland, Ohio 44135

Nomenclature

A, B, C, D	matrices that describe system in state-space form
alt	altitude
C-MAPSS	Commercial Modular Aero-Propulsion System Simulation
DF_O	5-element vector that defines top-level fault distribution
DF_fan	3-element vector that defines fault distribution in fan
DF_HPC	3-element vector that defines fault distribution in high-pressure compressor
DF_HPT	2-element vector that defines fault distribution in high-pressure turbine
DF_LPC	3-element vector that defines fault distribution in low-pressure compressor
DF_LPT	2-element vector that defines fault distribution in low-pressure turbine
FC	flight condition
GUI	graphical user interface
HPC	high-pressure compressor
HPT	high-pressure turbine
K(s)	incremental part of controller
LEM	linear engine model
LPC	low-pressure compressor
LPT	low-pressure turbine
N1	fan spool speed
N2	core spool speed
Q(s)	desired closed-loop transfer function
rps	radians per second
TRA	throttle-resolver angle
Tsl	sea level temperature
ω_n	undamped angular natural frequency
ζ	damping ratio

1.0 Introduction to C-MAPSS

1.1 Background

C-MAPSS stands for 'Commercial Modular Aero-Propulsion System Simulation' and it is a tool for the simulation of a realistic large commercial turbofan engine. The code is a combination of Matlab (The MathWorks, Inc.) and Simulink (The MathWorks, Inc.) with a number of graphical user interface (GUI) screens that allow point-and-click operation and with editable fields that allow the user to enter specific values of his/her own choice. In addition to the engine model (called the 90K because it is produces about 90,000 lb of thrust), the package includes an atmospheric model capable of operation at (i) altitudes from sea level to 40,000 ft, (ii) Mach numbers from 0 to 0.90, and (iii) sea-level temperatures from –60 to 103 °F. The package also includes a power-management system that allows the engine to be operated over a wide range of thrust levels throughout the full range of flight conditions.

A comprehensive control system is included that consists of (i) a fan-speed controller for which the user specifies the throttle-resolver angle (TRA), (ii) three high-limit regulators that prevent the engine from exceeding its design limits for core speed, engine-pressure ratio, and HPT exit temperature, (iii) a fourth limit regulator that prevents the static pressure at the HPC exit from going too low, (iv) acceleration and deceleration limiters for the core speed, and (v) a comprehensive logic structure that integrates these control-system components in a manner similar to that used in real engine controllers such that integrator-windup problems are avoided. Furthermore, all of the gains for the fan-speed controller and the four limit regulators are scheduled such that the controller and regulators perform as intended over the full range of flight conditions and power levels. The engine diagram in Figure 1.1 shows the main elements of the engine model and the flow chart in Figure 1.2 shows how the various subroutines are assembled in the simulation.

A number of GUI screens have been developed that make it easy for the user to work with either the open-loop engine (without any controller) or with the engine and its control system (closed loop). For the open-loop engine, transient simulations to doublet inputs can be run and linear engine models (LEMs) can be developed that have 14 inputs (Table 1.1) and 27 outputs (Table 1.2). C-MAPSS variables that are currently available internally but are not among the output variables are listed in Table 1.3. The inputs are fuel flow and a set of 13 health-parameter inputs that allow the user to simulate the effects of faults and deterioration in any of the engine's five rotating components (fan, LPC, HPC, HPT, and LPT). Using the GUIs provided for open-loop analysis, it is a simple matter for the user to save the LEM for later use and to compare its response with that of the nonlinear engine.

A controller-design GUI guides the user in the design of fan-speed controllers and limit regulators, using a LEM to represent the engine. The design GUI implements the model-matching algorithm of John Edmunds (ref. 1). However, it can be adapted for use with other design methods, should the user wish to do so. In order to avoid model complexity that is not required unless controllers are being designed that are capable of running a real engine, and to be able to attain fast execution speeds, the sensors and actuators are assumed to be ideal. By this, we mean that they have no dynamics, no computational time delays, and no errors or biases. Hence, they are treated as unity gains and do not appear in the model. The inclusion of non-ideal sensor and actuator models is discussed in Section 4.4.

Several GUIs are available that make it easy for the user to simulate the response of the engine and its control system in a variety of situations. A popup menu is included that contains 14 predefined flight conditions that cover a wide range of altitudes, Mach numbers, and power levels (Table 1.4). Also, the user is free to create his/her own flight conditions and save them in binary files for later use.

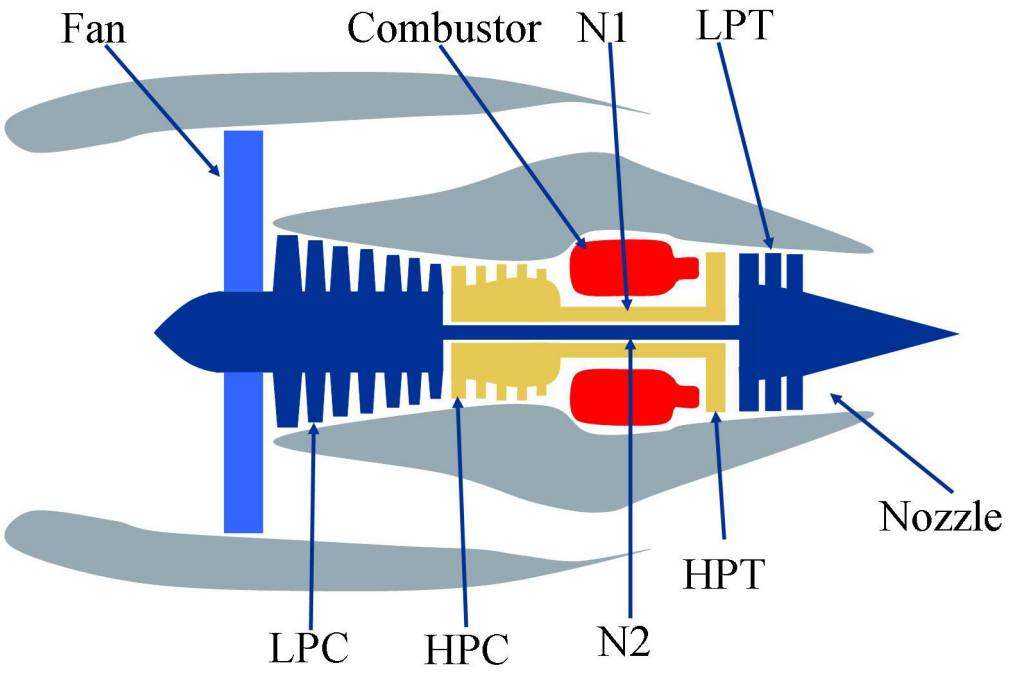

Figure 1.1.—Simplified diagram of the 90K engine.

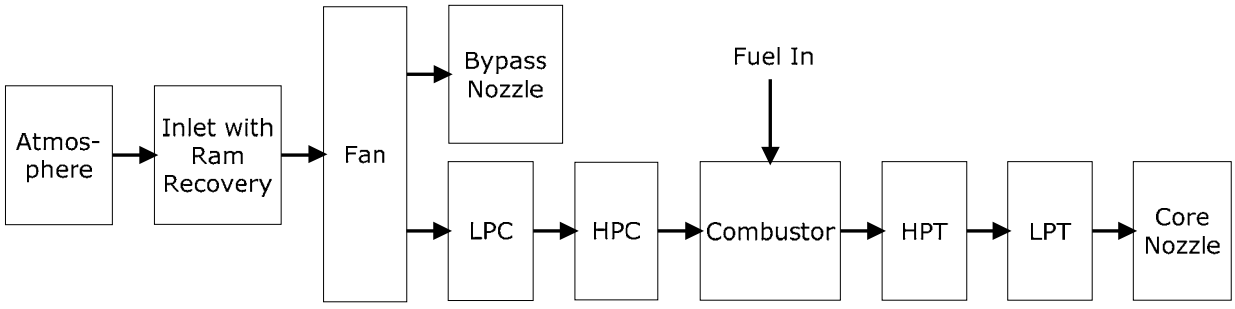

Figure 1.2.—Subroutines of the 90K engine simulation with ducts and bleed omitted.

TABLE 1.1.—INDEX, NAME, AND SYMBOL OF
14 INPUTS TO THE 90K ENGINE MODEL.

Index	Name	Symbol
1	Fuel flow	Wf (pps)
2	Fan efficiency modifier	fan_eff_mod
3	Fan flow modifier	fan_flow_mod
4	Fan pressure-ratio modifier	fan_PR_mod
5	LPC efficiency modifier	LPC_eff_mod
6	LPC flow modifier	LPC_flow_mod
7	LPC pressure-ratio modifier	LPC_PR_mod
8	HPC efficiency modifier	HPC_eff_mod
9	HPC flow modifier	HPC_flow_mod
10	HPC pressure-ratio modifier	HPC_PR_mod
11	HPT efficiency modifier	HPT_eff_mod
12	HPT flow modifier	HPT_flow_mod
13	LPT efficiency modifier	LPT_eff_mod
14	HPT flow modifier	LPT_flow_mod

TABLE 1.2.—LIST OF 27 OUTPUT VARIABLES, WITH THEIR
INDICES IN THE OUTPUT VECTOR *y* AND THEIR UNITS.

Index	Symbol	Description	Units
1	Nf	Physical fan speed	rpm
2	Nc	Physical core speed	rpm
3	epr	Engine pressure ratio (P50/P2)	--
4	P21	Total pressure at fan outlet	psia
5	T21	Total temperature at fan outlet	°R
6	P24	Total pressure at LPC outlet	psia
7	T24	Total temperature at LPC outlet	°R
8	P30	Total pressure at HPC outlet	psia
9	T30	Total temperature at HPC outlet	°R
10	P40	Total pressure at burner outlet	psia
11	T40	Total temperature at burner outlet	°R
12	P45	Total pressure at HPT outlet	psia
13	T48	Total temperature at HPT outlet	°R
14	P50	Total pressure at LPT outlet	psia
15	T50	Total temperature at LPT outlet	°R
16	W21	Fan flow	pps
17	Fn	Net thrust	lbf
18	Fg	Gross thrust	lbf
19	SmFan	Fan stall margin	--
20	SmLPC	LPC stall margin	--
21	SmHPC	HPC stall margin	--
22	NRf	Corrected fan speed	rpm
23	NRc	Corrected core speed	rpm
24	P15	Total pressure in bypass-duct	psia
25	PCNfR	Percent corrected fan speed	pct
26	Ps30	Static pressure at HPC outlet	psia
27	phi	Ratio of fuel flow to Ps30	pps/psi

TABLE 1.3.—NON-OUTPUT VARIABLES WRITTEN TO THE WORKSPACE

Symbol	Description	Units
accel_in	Accel limiter input	rpm/s
accel_out	Accel limiter output	rpm/s
BPR	Bypass ratio	---
DD	Decel limiter output	rpm/s
farB	Burner fuel-air ratio	---
far_HPT	HPT fuel-air ratio	---
far_LPT	LPT fuel-air ratio	---
Fdrag	Drag force	lbf
htBleed	Bleed enthalpy	
Nf_dot	Fan acceleration	rpm/s
Nc_dot	Core acceleration	rpm/s
Nf_dmd	Demanded fan speed	rpm
P2	Pressure at fan inlet	psia
PCNfRdmd	Demanded corrected fan speed	pct
PCNfR_filtered	Output of pcnfr filter for gain scheduling	pct
PR_HPC	Pressure ratio of HPC	---
PR_HPT	Pressure ratio of HPT	---
PR_LPT	Pressure ratio of LPT	---
tau_HPC	Torque of HPC	ft-lb
tau_HPT	Torque of HPT	ft-lb
tau_LPT	Torque of LPT	ft-lb
TRA	Throttle resolver angle	deg
T2	Total temperature at fan inlet	°R
W22	Flow out of LPC	lbm/s
W25	Flow into HPC	lbm/s
W31	HPT coolant bleed	lbm/s
W32	HPT coolant bleed	lbm/s
W48	Flow out of HPT	lbm/s
W50	Flow out of LPT	lbm/s
Wf_dot	Derivative of fuel flow	lbm/s^2
x1,…,x5	Solver outputs	

TABLE 1.4.—EQUILIBRIUM VALUES FOR THE 14 FLIGHT CONDITIONS IN THE MENU

[Note: sea-level temperature is 59 °F (standard day) for all flight conditions, except FC03, for which it is 86 °F]

Name	Alt, ft	Mach	Tsl, °F	TRA, deg	Fuel flow, pps	Fan speed, rpm	Core speed, rpm	epr	HPT outlet temp, °R	Net Thrust, lbf
FC01	0	0	59	100	6.835	2388	9051	1.300	2072	86,336
FC02	0	0.25	59	100	7.085	2403	9084	1.261	2083	66,755
FC03	0	0.25	86	96	7.043	2432	9274	1.247	2162	64,250
FC04	1000	0	59	100	6.567	2380	9021	1.300	2059	83,293
FC05	10 K	0.25	59	100	4.661	2319	8774	1.259	1947	45,830
FC06	20 K	0.70	59	100	3.863	2324	8719	1.077	1909	25,774
FC07	25 K	0.62	59	60	1.670	1915	8006	0.938	1534	11,475
FC08	35 K	0.84	59	100	2.120	2223	8346	1.024	1750	13,552
FC09	42 K	0.84	59	100	1.518	2212	8317	1.023	1744	9,647
FC10	0	0	59	80	5.511	2224	8837	1.227	1941	71,652
FC11	0	0	59	60	4.254	2028	8592	1.165	1792	57,181
FC12	0	0	59	40	3.075	1797	8299	1.114	1623	42,562
FC13	0	0	59	20	2.013	1497	7946	1.073	1433	28,016
FC14	0	0	59	0	1.123	1146	7503	1.044	1214	13,448

1.2 Version Note

This guide is intended to be used with the initial release of the C-MAPSS software (version 1.0). The tutorials throughout this guide have been generated using Matlab R14 Service Pack 2 (Matlab v. 7.0.4, Simulink v. 6.2) under the Windows operating system. It is expected that C-MAPSS will be able to run on any Matlab version and any Matlab-supported operating system (e.g., Linux (Linus Torvalds), Mac OS (Apple, Inc.)), although this has not been completely verified. On operating systems other than Windows, the simulation will not run unless the executable code included with the package (DLL files) are recompiled from the source C-code to generate platform-specific executables. Procedures for doing this are discussed in more detail in Section 4.

1.3 Installation

C-MAPSS requires that Matlab/Simulink and the Control System Toolbox (preferably R14 SP2 or later) are installed on the machine. In order to install C-MAPSS, it is only necessary to place the `c-mapss` top-level folder in a directory accessible by Matlab. In order to be a valid Matlab directory, there must be no spaces anywhere in the hierarchy (e.g., '`My Documents`'). This is to avoid errors upon setting the Matlab path. On Linux and Mac platforms, directory structures utilize only forward slashes (`/`), therefore it is necessary to replace all of the backslashes (`\`) in the C-MAPSS path setup routine before C-MAPSS can be used on these platforms. To do this, open `setup_path_etc.m` in the `/c-mapss` top-level directory and replace the backslashes following each `addpath` command with forward slashes. Performing a find/replace on the contents of the file should make quick work of this.

2.0 Using the GUI Features

The TOP-LEVEL GUI provides the user with three major functions: simulate and linearize the open loop engine, design controllers, and simulate the closed loop engine. The GUI is shown in Figure 2.1. The procedure for starting the simulation at the beginning of a Matlab session is to first navigate to the `\c_mapss` directory, then type the command `setup_everything` at the command line prompt. This will initialize the Matlab paths, load the engine parameters, and open the TOP-LEVEL GUI. When using the simulation GUIs, it is recommended that the command window be visible, as this displays important information that results from user action.

2.1 Simulation of the Open-Loop Engine and Generation of Linear Models

The OPEN-LOOP ANALYSIS AND LINEARIZATION GUI (Figure 2.2), accessed by clicking SIMULATE & LINEARIZE in the TOP-LEVEL GUI, allows the user to operate the engine simulation to perform the following functions:

1) Simulate the open-loop response to a doublet in any of its 14 inputs (i.e., a small step increase from the starting value, followed by a symmetrical decrease below the starting value, followed by a return to the starting value).
2) (a) create a linearized model (LEM), (b) compare the response of the LEM with that of the nonlinear engine, (c) save the LEM as a binary file for use with the design GUI, and (d) load a previously-saved LEM.

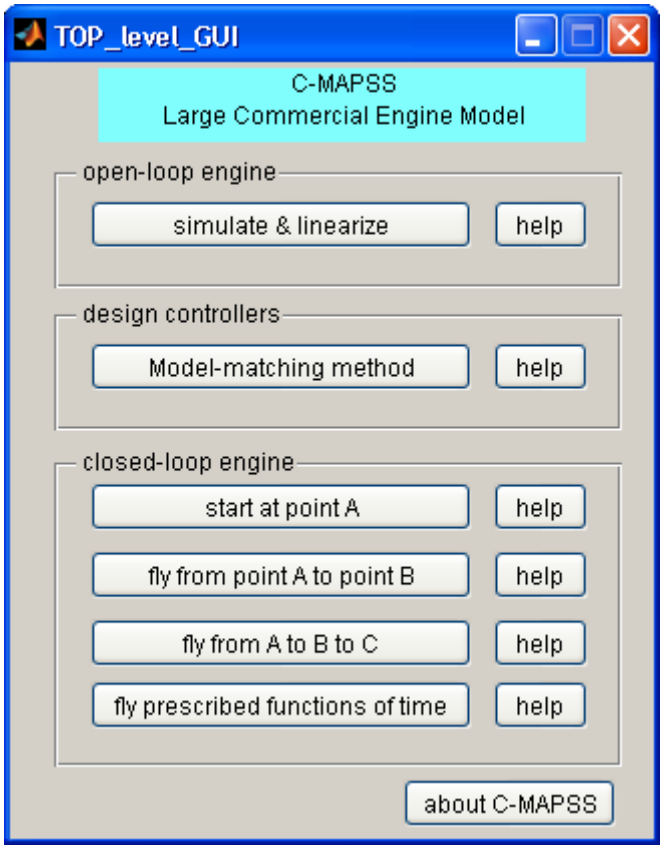

Figure 2.1.—Top-level C-MAPSS GUI.

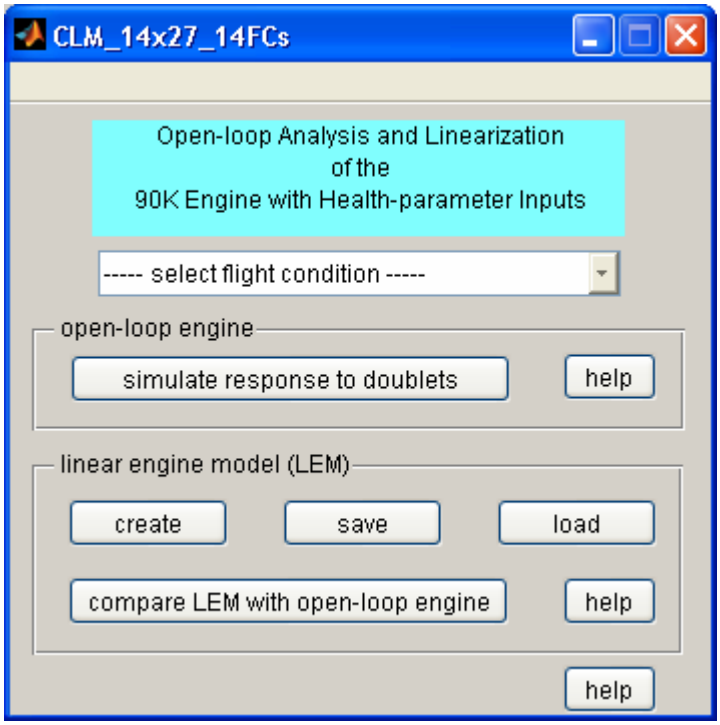

Figure 2.2.—GUI for open-loop engine activities.

2.1.1 Open-Loop Engine

First, the user must select one of the 14 flight conditions from the popup menu. Then, to simulate the doublet response of the engine, click on the button named SIMULATE RESPONSE TO DOUBLETS in the OPEN-LOOP ANALYSIS AND LINEARIZATION GUI. Doing so will cause the GUI shown in Figure 2.3 to appear. It allows the user to simulate the response of the open-loop engine to a total of 14 inputs, namely the fuel flow and the 13 health parameters.

In the upper panel, labeled SETUP & RUN, click on the popup menu and select one of the 14 inputs, where fuel flow is the first, and the other 13 are the engine health parameter inputs (fan-efficiency modifier, fan-flow modifier, etc). Then click on the button labeled RUN SIMULATION. This will bring up the proper Simulink model diagram and run the simulation for 10 sec. When the simulation has finished, the user can generate response plots by clicking on any of the six buttons in the plots panel.

2.1.2 Linear Engine Model (LEM)

In the OPEN-LOOP ANALYSIS AND LINEARIZATION GUI, make sure a selection has been made in the flight conditions popup menu. The CREATE button will bring up a Simulink model that is run automatically to compute the necessary partial derivatives and compute the corresponding LEM in state-space form. Clicking the SAVE button will bring up a GUI that has an edit field for specifying the name of the binary .MAT file to be saved with the LEM that has just been computed. Clicking the LOAD button will bring up a GUI that will allow the user to specify the name of a previously-saved LEM file. Clicking the COMPARE LEM WITH OPEN-LOOP ENGINE button will bring up the GUI shown in Figure 2.4.

First, the input to be used for the comparison must be specified via the popup menu in the upper panel, which is named SETUP & RUN. Then the button named RUN SIMULATION should be clicked. This will bring up a Simulink model that will start automatically and run for 10 sec. Once the simulation has finished, the buttons in the lower panel labeled PLOTS can be used. In the plots, the engine's responses are solid blue lines and the LEM's responses are dashed red lines.

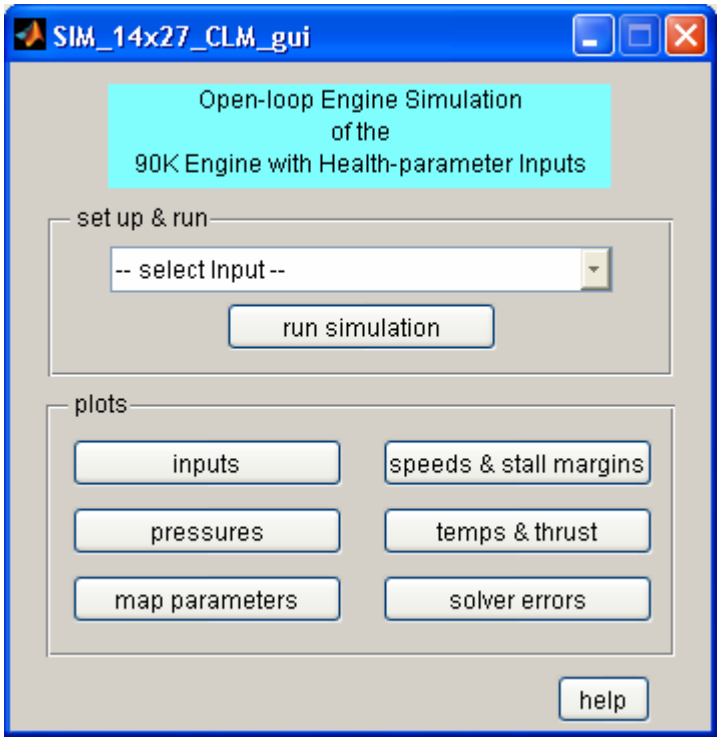

Figure 2.3.—GUI for simulation of the open-loop engine.

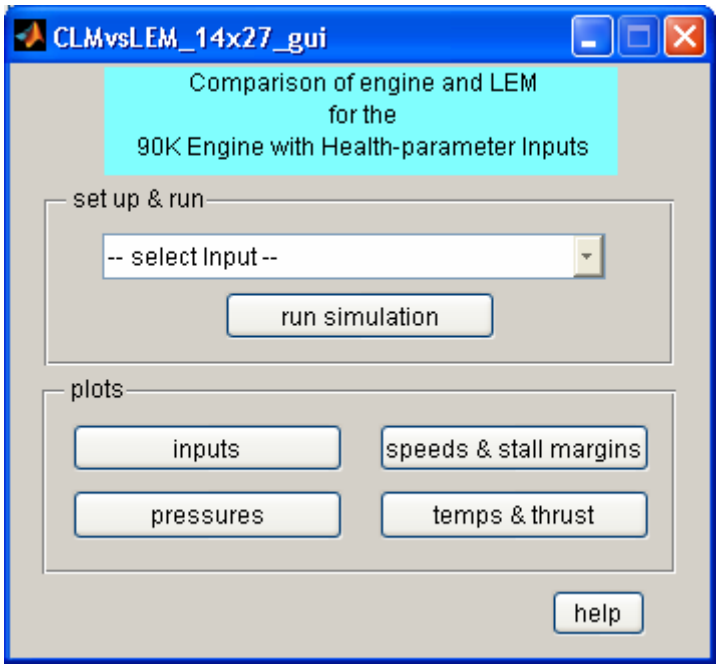

Figure 2.4– GUI for comparing the LEM with the open-loop engine.

2.2 Controller Design With the Model-Matching Algorithm

The REGULATOR DESIGN GUI (Figure 2.5), accessed by clicking MODEL-MATCHING METHOD in the TOP-LEVEL GUI allows the user to design controllers and limit regulators using the model-matching algorithm described in reference 1. Briefly, the user creates a 2nd-order system, referred to as Q(s), whose response represents the desired response of the closed-loop engine. This system is expressed in terms of its undamped natural frequency ω_n (rps) and its damping ratio ζ, which are specified in the upper two editable windows in the GUI.

The controller is divided into two parts (Figure 2.6):

1) the incremental part, referred to as K(s), which is designed with this GUI, and is of order one (scalar gain, single pole, and single zero), and
2) the free integrator, which is considered to be part of the plant for the design calculations

The user assigns the magnitude of the single pole of the incremental part of the controller's transfer function in the lower editable window. The controller produced by the design calculations gives the best fit, in a least-squares sense, of the actual closed-loop frequency response (engine + K(s) + free integrator) to the frequency response of the desired closed system (Q(s)).

2.2.1 Design the Controller

First, the user must select one of the 14 flight conditions from the popup menu (Figure 2.5). Next, the controller to be designed is specified via the SELECT REGULATOR popup menu. Then the button labeled BUILD DESIGN PLANT is clicked. This will cause the 2nd-order, single-input, single-output design plant, exclusive of the free integrator, to be displayed in state-space form in the Matlab command window, along with the DC gains of the scaled and unscaled linear engine models. If desired, the values of the three design parameters can be changed in the respective editable fields, but the default values of 4.0, 0.70, and 20 rps should prove satisfactory. Once the button labeled ACCEPT DESIGN PARAMETERS has

been clicked, the button labeled DO THE DESIGN should be clicked. The design can be saved as a binary file by clicking the SAVE K(S) button, which will bring up a small GUI in which the user can specify the name of the file.

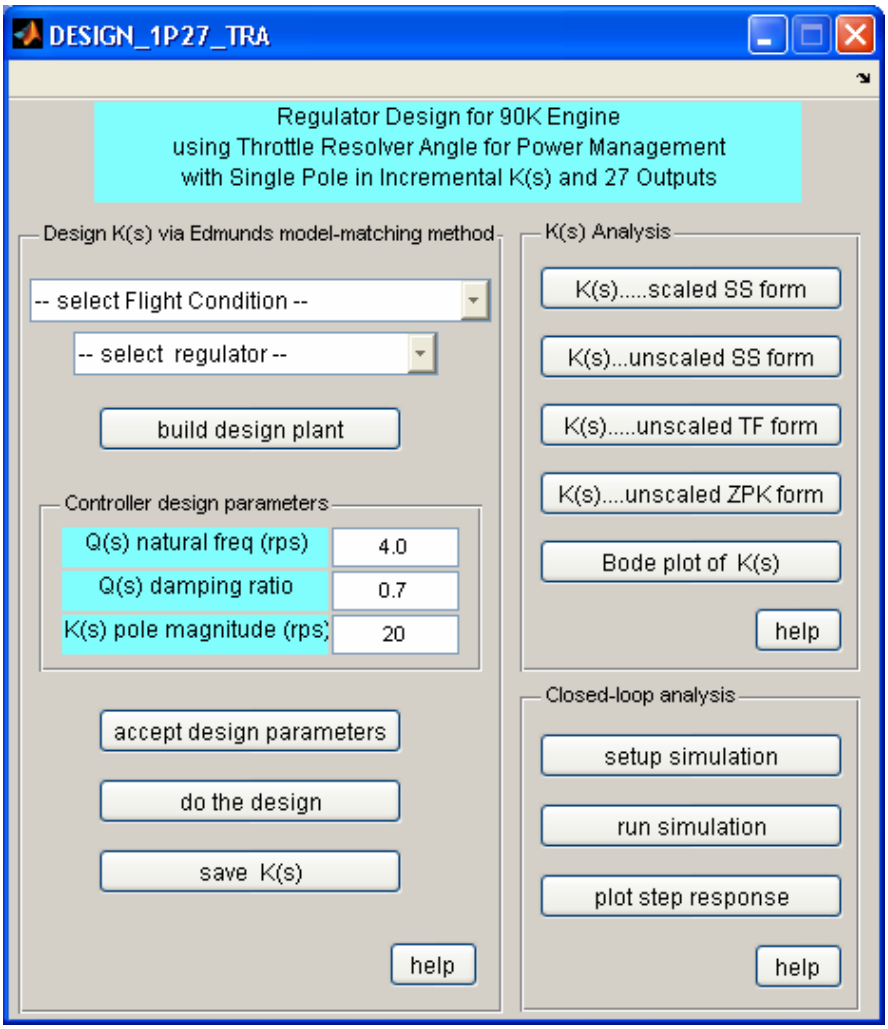

Figure 2.5.—GUI for designing controllers.

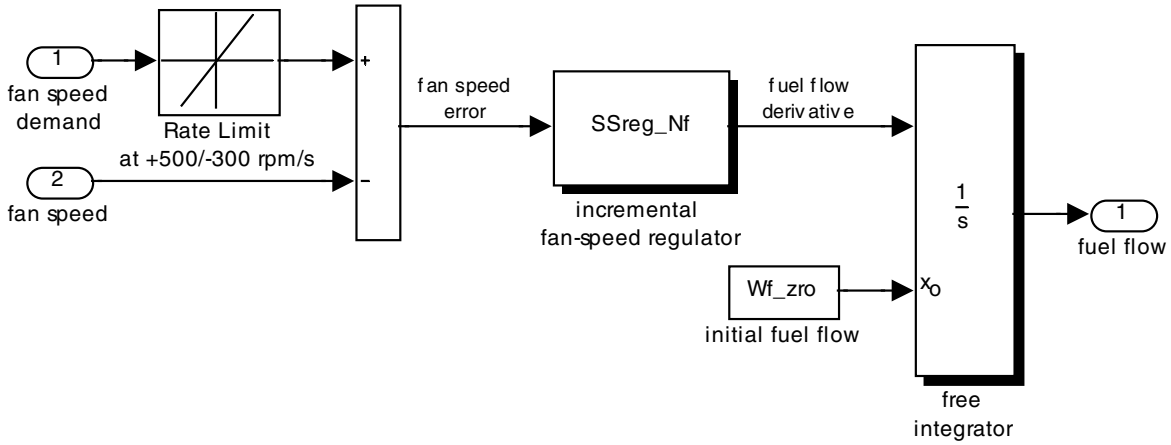

Figure 2.6.—Fan-speed controller, consisting of incremental controller and free integrator.

2.2.2 Controller Analysis

The first four of these buttons (top right of Figure 2.5) cause the incremental part of the designed controller to be displayed in the Matlab command window. The first of these shows the scaled controller in state-space form, whereas the other three buttons show the controller in unscaled form. The design calculations are done using the scaled form, but the unscaled form must be used to control the nonlinear engine model. The first of these is the state-space form (matrices A, B, C, and D) of the unscaled controller. The second is the transfer function, expressed in terms of a gain and the coefficients of the numerator and denominator polynomials. The third option is the transfer function expressed in terms of its gain, zero, and pole. Detailed explanations of the two transfer-function forms can be seen in the command window.

If the user clicks the fifth button, the Bode plot of the incremental part of the scaled controller (magnitude in dB and phase in degrees, versus frequency in radians per second) is displayed. The free-integrator portion of the controller is not included in the plot.

2.2.3 Closed-Loop Analysis

These three buttons (bottom right of Figure 2.5) allow the user to simulate the step response of the closed-loop system consisting of the incremental controller (K(s)), the free integrator (1/s), and the linear engine model (LEM). When the user clicks on the first button, a Simulink model will appear that shows the individual components connected as a feedback loop, with a step-function input (Figure 2.7). All components are in scaled form. The second button causes the simulation to be run, after which the third button can be used to produce a plot of the controlled variable (fan speed, core speed, epr, T48, or Ps30) and the fuel flow. The plots show the variables in terms of their unscaled values. The base values shown on the plot are the equilibrium values for the flight condition for which the design has been performed.

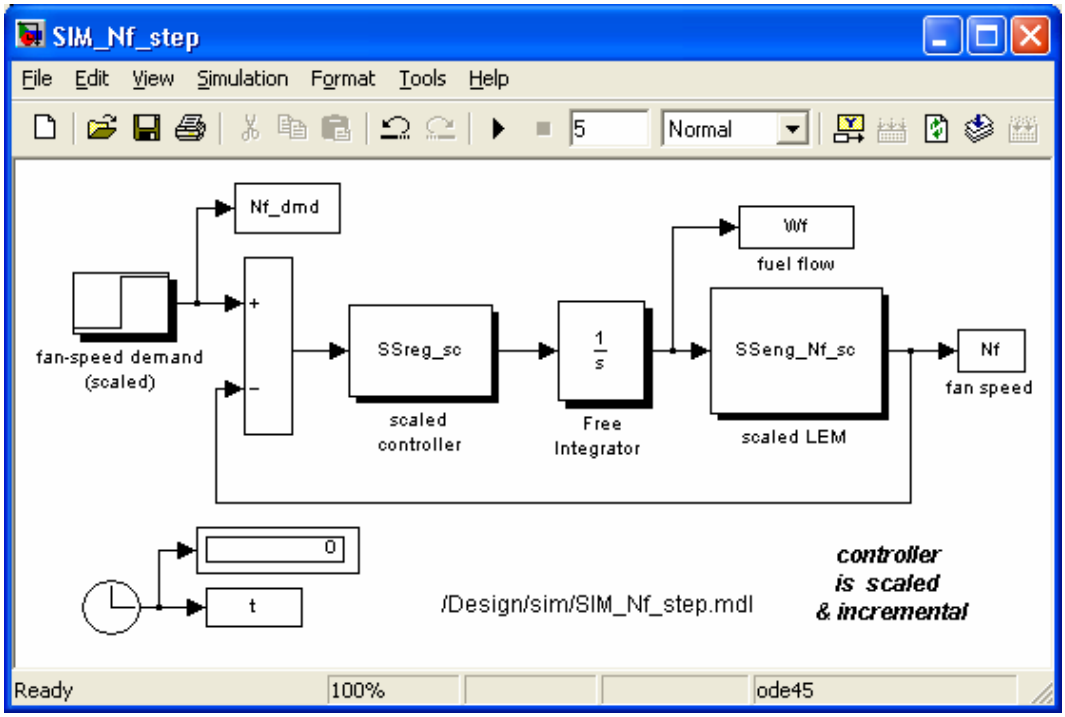

Figure 2.7.—Closed-loop system with step input.

2.3 Simulation of the Controlled Engine

2.3.1 Flying the Closed-Loop Engine With a Step Change in TRA, Starting at Point A

Clicking START AT POINT A in the TOP-LEVEL GUI (Figure 2.1) brings up the GUI shown in Figure 2.8. This GUI allows the user to operate with the closed-loop engine in one of three configurations and start at any of the 14 flight conditions in the popup menu (this is Point A) and simulate the response to a step change in TRA.

The three configurations are:

1) Only fan-speed control, with point gains
2) Only fan-speed control, with scheduled gains
3) Fan-speed control and four limit regulators, with all gains scheduled.

The user must first select one of the 14 flight conditions from the popup menu.

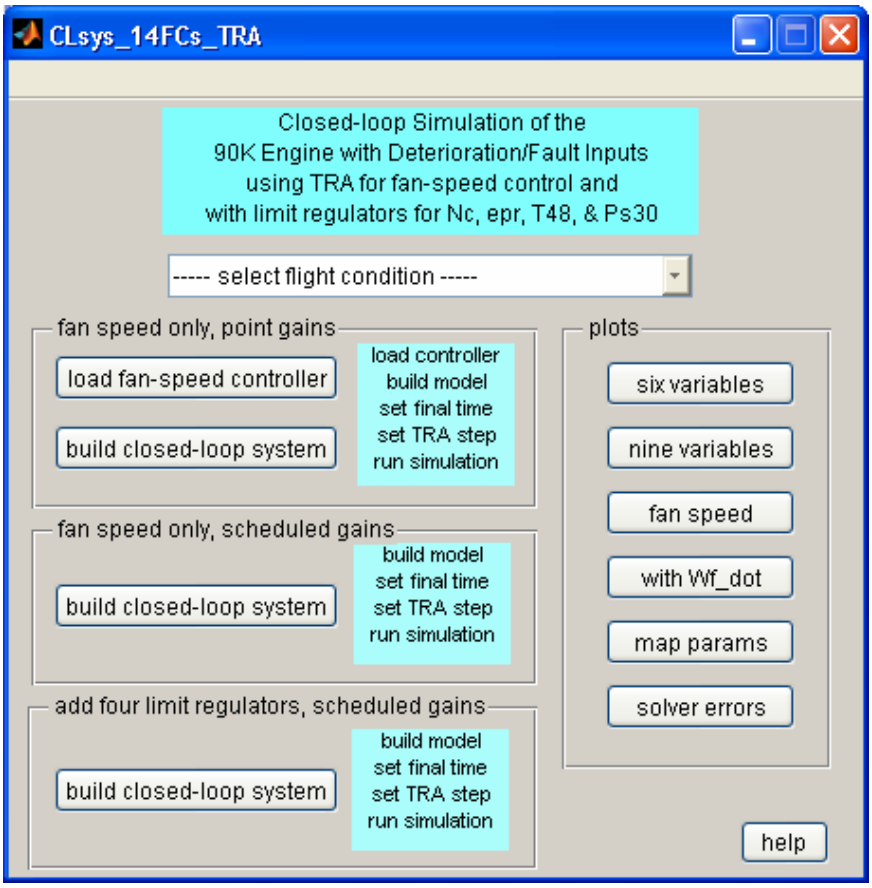

Figure 2.8.—GUI for simulating response starting at Point A.

2.3.1.1 Fan-Speed Control Only, Point Gains

Click the LOAD FAN-SPEED CONTROLLER button. This will cause a GUI to appear along with a list of .MAT files of previously generated controllers. The GUI allows the user to specify the .MAT file for the controller from the list by typing its name, then pressing LOAD. Alternately, the user may select a .MAT file from the "Current Directory" window. Several default controllers corresponding to the 14 default flight conditions have been included with this C-MAPSS release. Next, click the BUILD CLOSED-LOOP SYSTEM button. After the Simulink model shown in Figure 2.9 has appeared, the user should set the sign and magnitude of the step in TRA (shaded block), set the final time, and run the simulation. TRA can be set to any value in the range between 0 (minimum power) and 100° (maximum power). After the simulation has completed running, any of the six buttons in the plot panel (right side of Figure 2.8) can be used for plots.

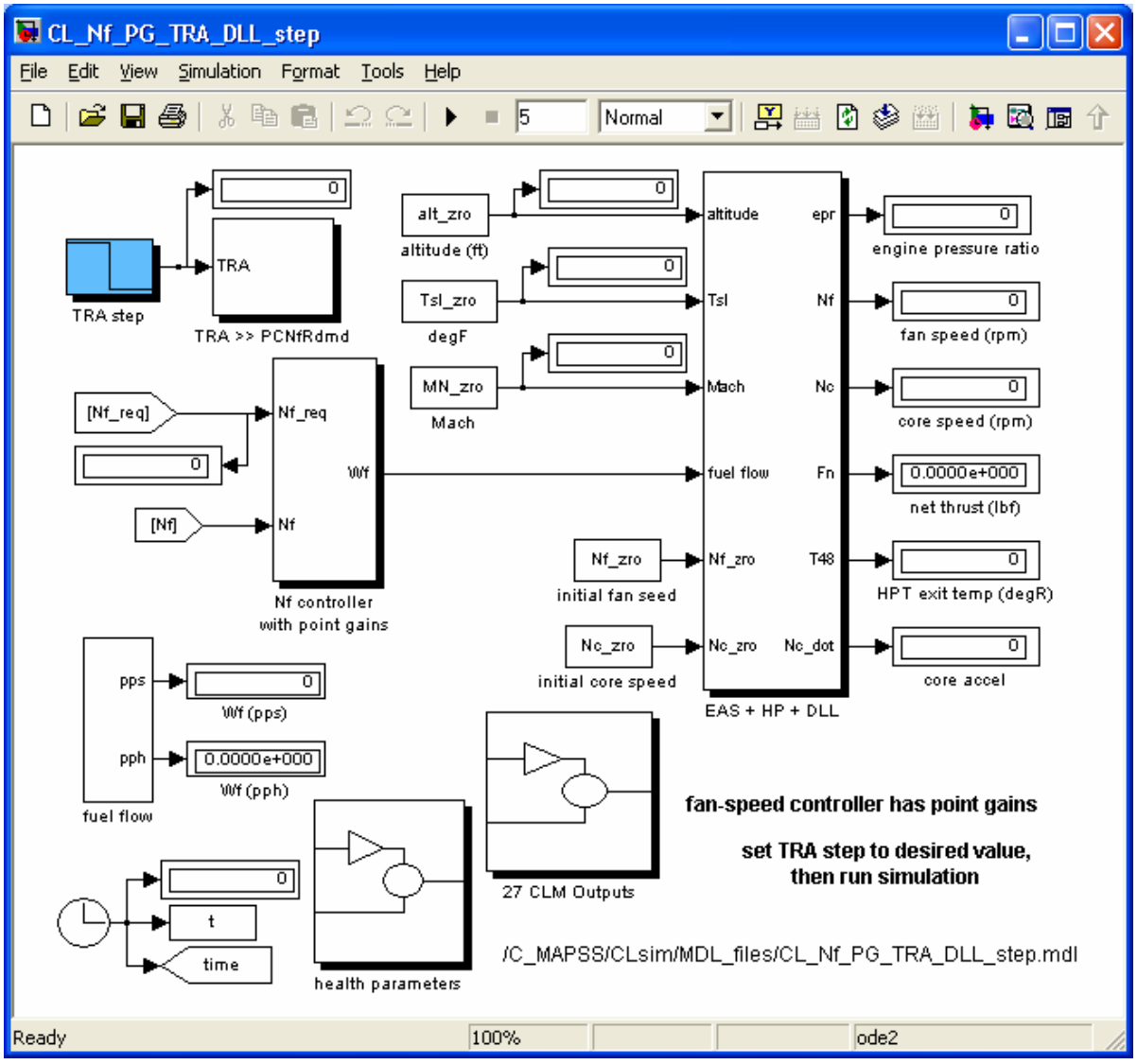

Figure 2.9.—Closed-loop system with fan-speed controller having point gains.

2.3.1.2 *Fan-Speed Control Only, Scheduled Gains*

The same steps as for the FAN-SPEED CONTROL ONLY, POINT GAINS option apply, except that it is not necessary to load the fan-speed controller. This is because the tables for the scheduled gains are loaded in the workspace at startup. In the Simulink model shown in Figure 2.10, the user should set the sign and magnitude of the TRA step (shaded block), set the final time, and run the simulation. TRA can be set to any value in the range between 0 (minimum power) and 100° (maximum power).

2.3.1.3 *Fan-Speed Control and Four Limit Regulators, Scheduled Gains*

When using this option, the limit-regulator schedules and fan speed gain schedules are in the workspace, so it is not necessary to load the fan speed controller. In the Simulink model shown in Figure 2.11, the user should set the sign and size of the magnitude of the TRA step (shaded block), set the final time, and run the simulation. TRA can be set to any value in the range between 0 (minimum power) and 100° (maximum power).

2.3.1.4 *Plot Results*

When the simulation has completed, plots can be produced by using the six plot buttons (right side of Figure 2.8).

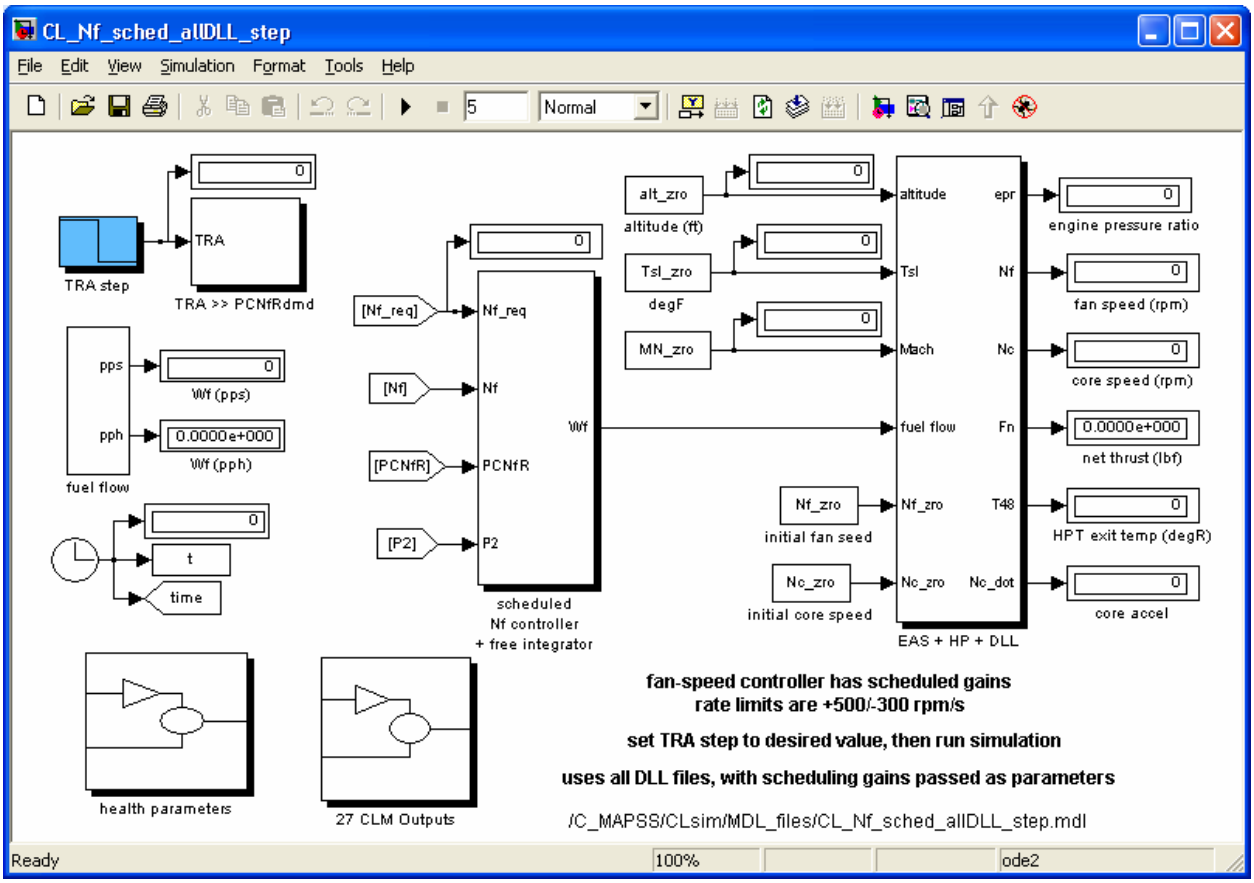

Figure 2.10.—Closed-loop system with fan-speed controller having scheduled gains.

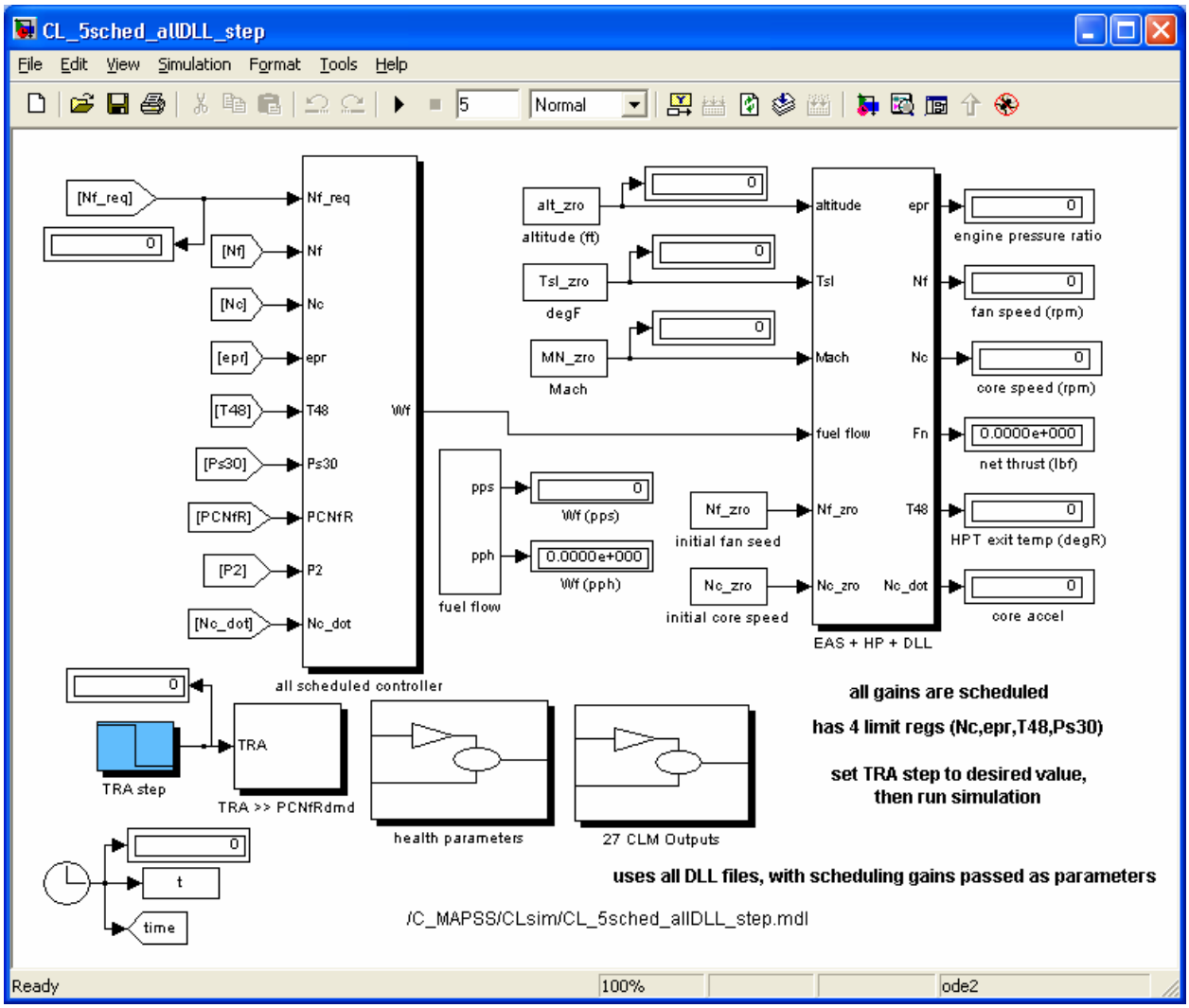

Figure 2.11.—Closed-loop system with fan-speed controller and four limit regulators, all with scheduled gains.

2.3.2 Flying the Closed-Loop Engine From Point A to Point B

This GUI (Figure 2.12) is accessed by clicking FLY FROM POINT A TO POINT B in the TOP-LEVEL GUI (Figure 2.1). Here, the user is able to fly the closed-loop engine from any of the 14 flight conditions in the popup menu (Point A) to another point (Point B) that the user defines. The engine will fly to Point B via a set of four ramp functions that make the user-defined transitions in altitude, Mach number, sea-level temperature, and TRA.

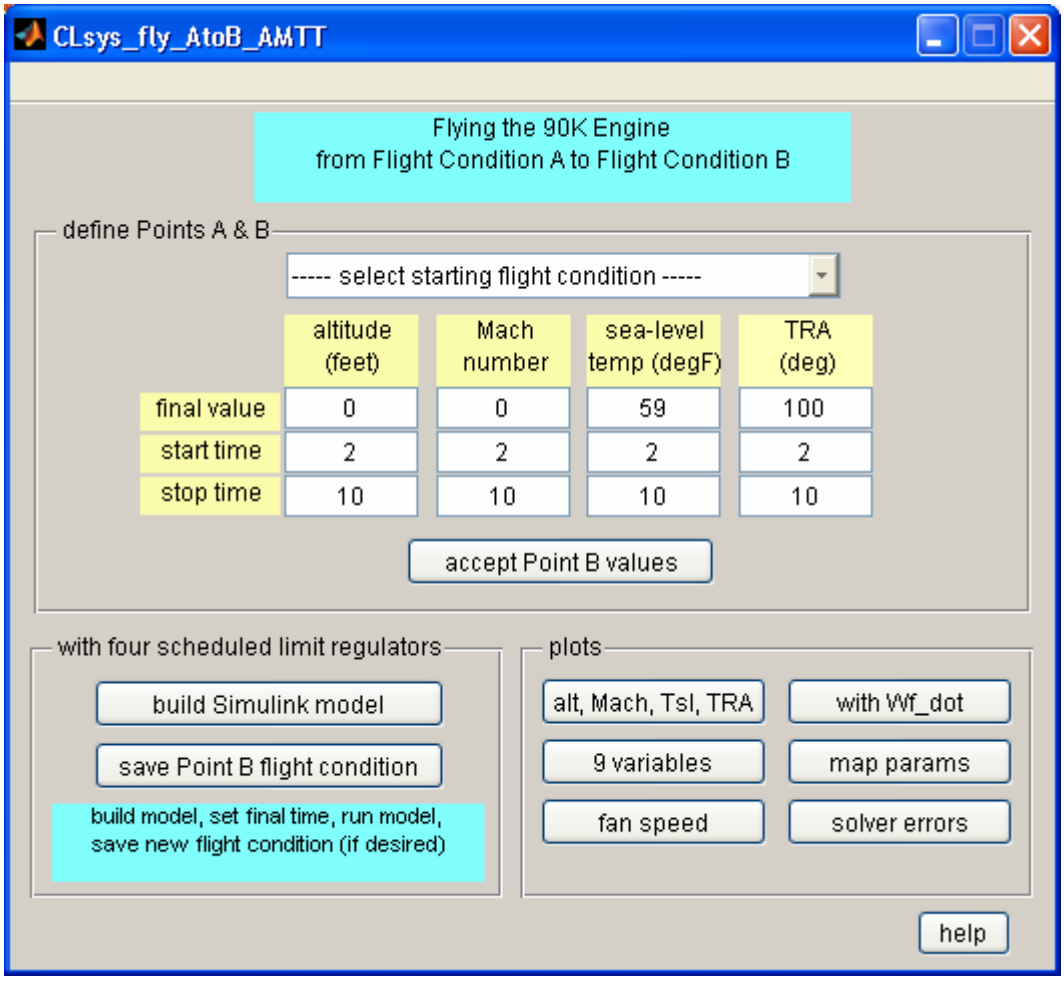

Figure 2.12.—GUI for flying from Point A to Point B.

2.3.2.1 Define Points A and B

First, the user must select one of the 14 flight conditions from the popup menu to define Point A. Then the four edit fields in the first row in the DEFINE POINTS A & B panel should be modified to define the altitude, Mach number, sea-level temperature, and TRA of Point B. Also, the start and stop times of the four ramps should be set in the second and third rows. When this has been done, the ACCEPT POINT B VALUES button should be clicked to complete the setup procedure.

2.3.2.2 Run Model with Four Scheduled Limit Regulators

Next, click the BUILD SIMULINK MODEL button in the lower left panel. This will cause the Simulink model in Figure 2.13 to appear, where the user can set the final time and run the model. When the simulation has completed, the conditions at Point B can be saved in a binary file for later use by clicking on the SAVE POINT B FLIGHT CONDITION button. The user will be prompted to enter the file name of the .MAT file where the data will be saved. The command window will display all flight condition files saved in the `\c_mapss\CLM\FC_files` directory.

2.3.2.3 Plot Results

When the simulation has completed, plots can be produced by using the six plot buttons.

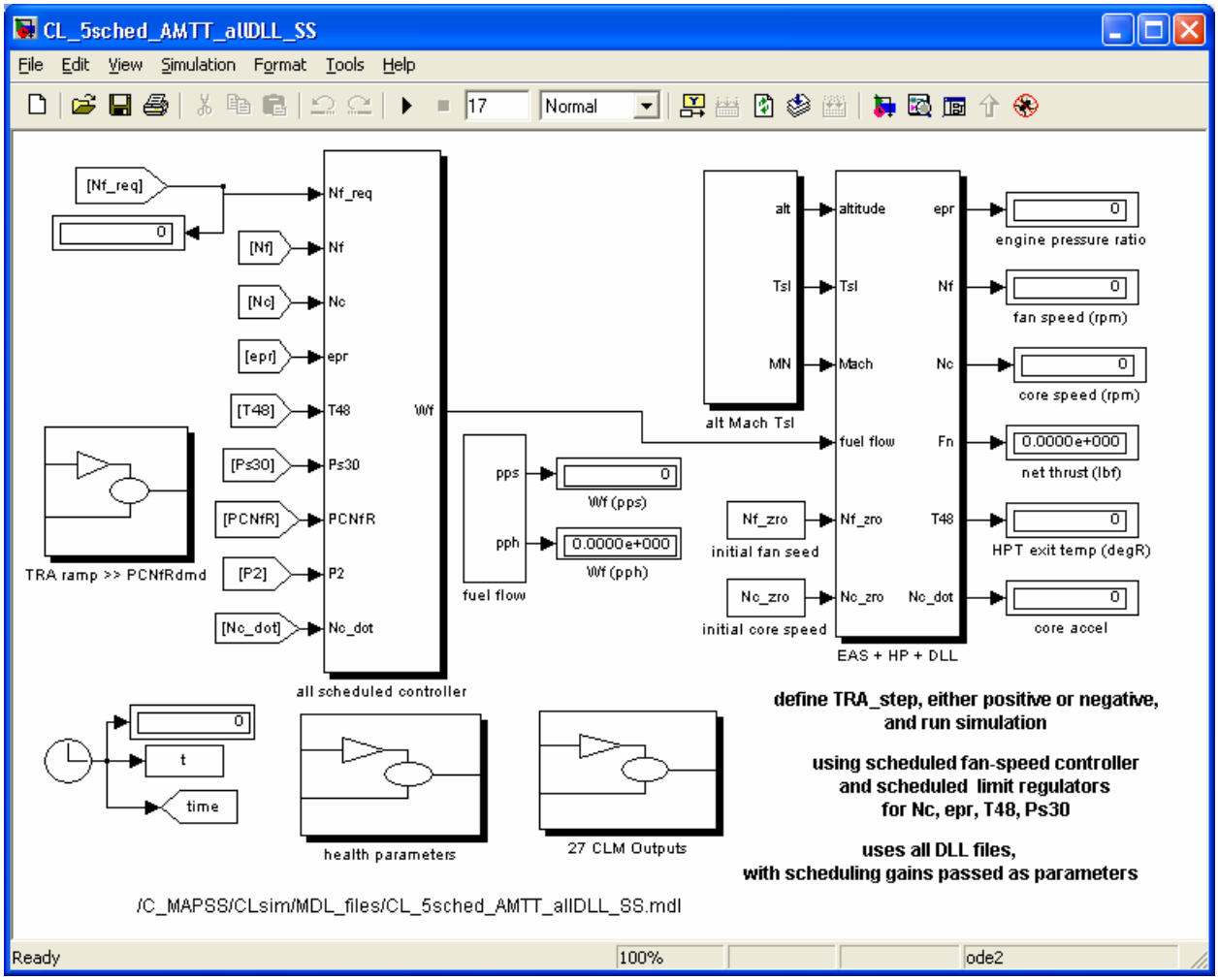

Figure 2.13.—Top-level of the model for flying from Point A to Point B.

2.3.3 Flying the Closed-Loop Engine From Point A to Point B to Point C

This GUI (Figure 2.14) is accessed by clicking FLY FROM A TO B TO C in the TOP-LEVEL GUI (Figure 2.1). This allows the user to fly the closed-loop engine from any of the 14 flight conditions in the popup menu (Point A) to a second point (Point B) that the user defines, hold at Point B for a user-defined interval, then fly to a third user-defined point (Point C), and hold at Point C until the conclusion of the run. The points B and C are defined in terms of altitude, Mach number, and TRA and are reached via user-defined ramps. The entire run is done at a constant sea-level temperature that the user can set (note that ambient temperature varies with altitude and Mach number). The user must enter start times that are monotonically increasing and the final time in the simulation must be greater than the 'hold at C' start time.

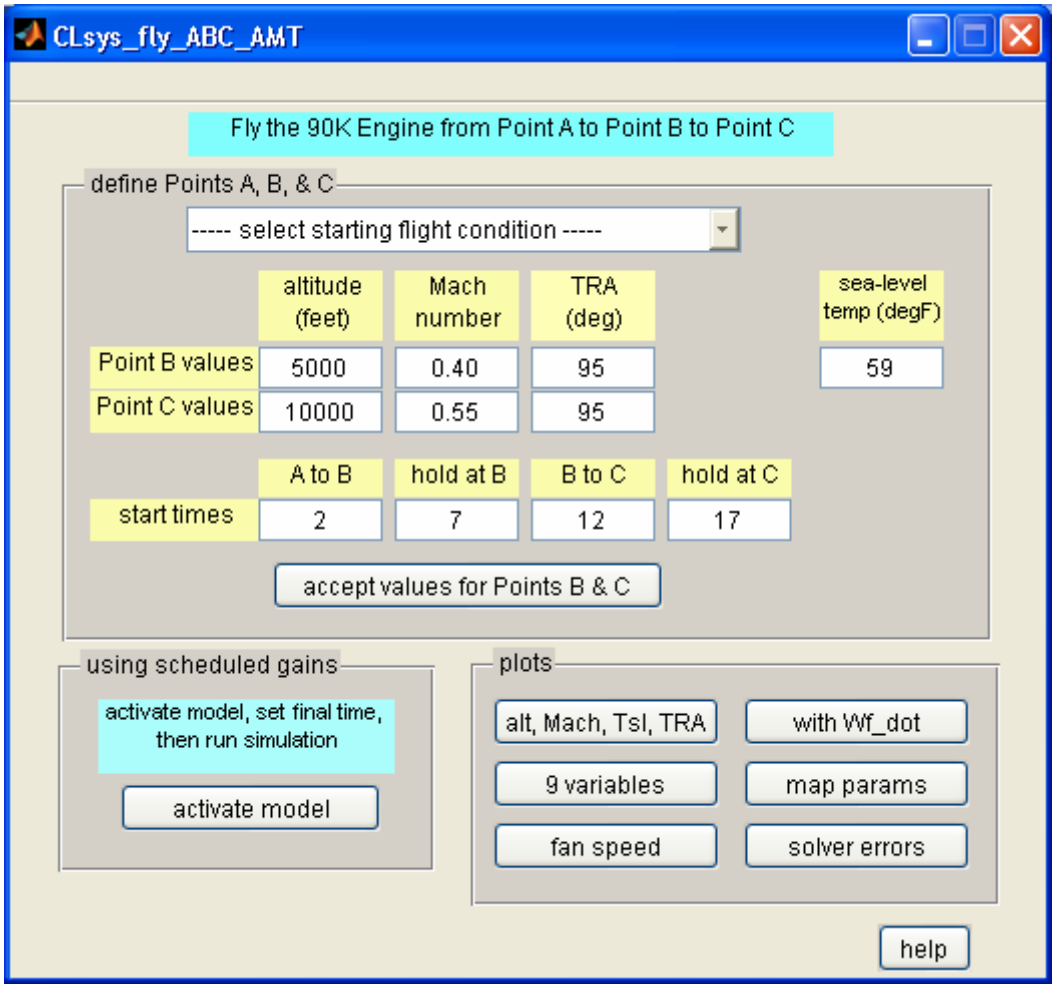

Figure 2.14.—GUI for flying from Point A to Point B to Point C.

2.3.3.1 Define Points A, B, & C

First, the user must select one of the 14 flight conditions from the popup menu to define Point A. Then the three edit fields in the first row in the DEFINE POINTS A, B, & C panel should be modified to define the altitude, Mach number, and TRA of Point B. Next, the corresponding values for Point C should be set in the second row of that panel. Then, the times at which each of the four phases is to start should be set in the third row of the panel. If the sea-level temperature is to be different from the default value of 59 °F (standard day), the field on the right side of the panel should be set to the desired value. Beware that values that differ from the temperature at the starting point (Point A) may cause startup transients or the model could even fail to run. When these steps have been done, click the button labeled ACCEPT VALUES FOR POINTS B & C at the bottom of the panel.

2.3.3.2 Run Using Scheduled Gains

To use the model, click the ACTIVATE MODEL button in the lower left panel. This will cause the following Simulink model to appear (Figure 2.15) and the user can set the final time and run the model.

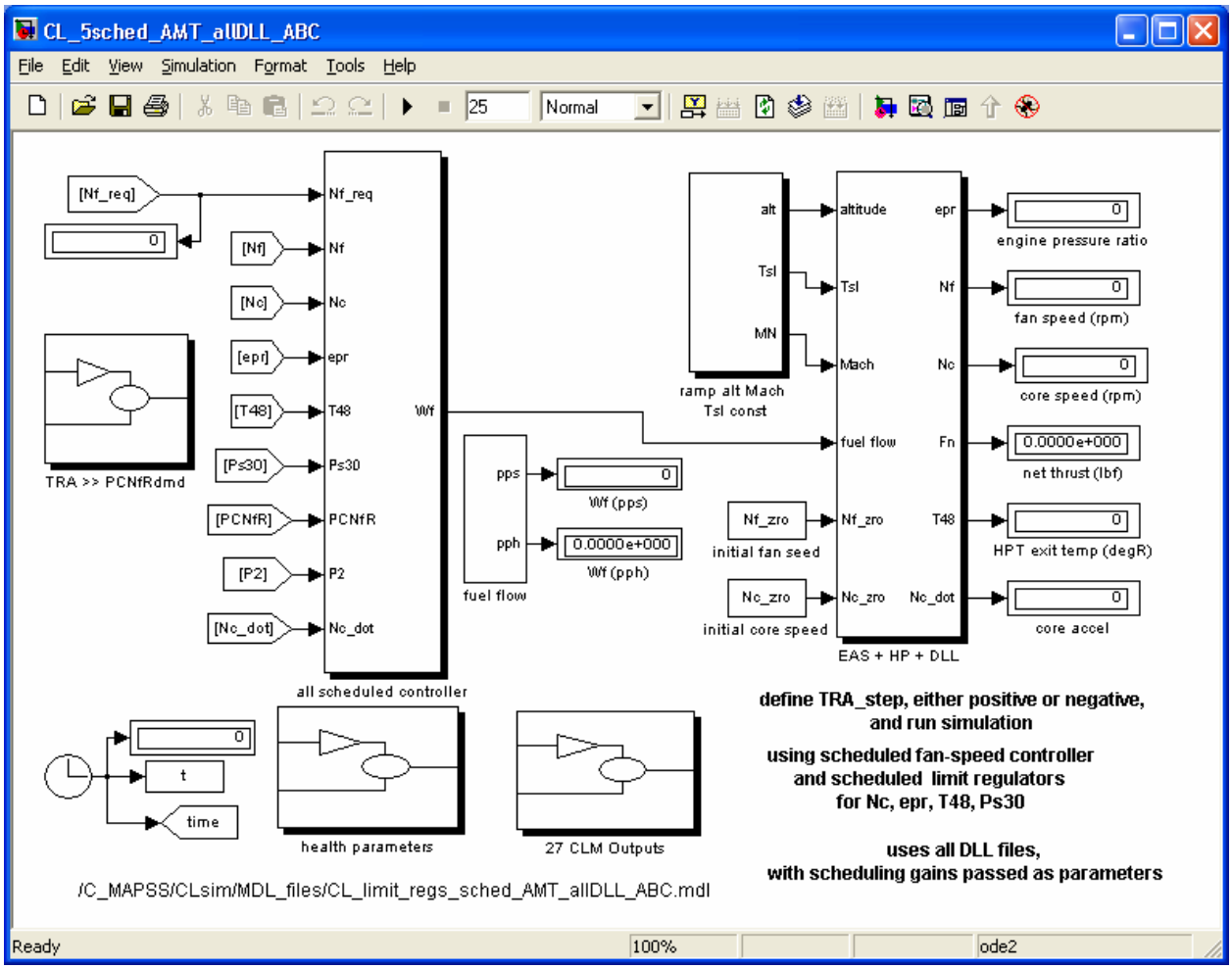

Figure 2.15.—Top-level of the model for flying from Point A to Point B to Point C.

2.3.3.3 Plot Results

When the simulation has completed, plots can be produced by using the six buttons in the plot panel.

2.3.4 Flying the Closed-Loop Engine Where Altitude, Mach Number and TRA are Prescribed Functions of Time

This GUI (Figure 2.16), accessed by clicking FLY PRESCRIBED FUNCTIONS OF TIME in the TOP-LEVEL GUI (Figure 2.1), allows the user to fly the closed-loop engine where altitude, Mach number and TRA are prescribed functions of time. The engine runs under fan-speed control with all four of the limit regulators active. All gains are scheduled, so the engine should be able to be run at whatever altitude, Mach number, and TRA the user specifies. Valid ranges are between zero and 40,000 ft altitude, zero to 0.9 Mach, and zero to 100° TRA. The full range of sea-level temperature (Tsl) values should work (–20 to 103 °F), provided the model is initialized at the desired temperature.

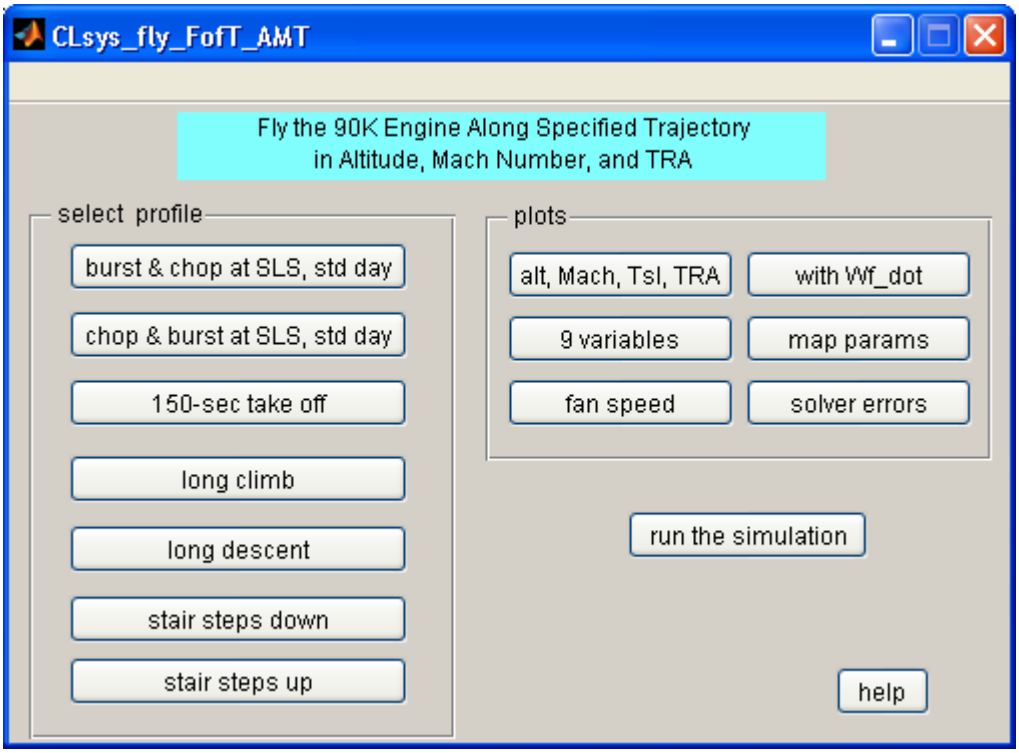

Figure 2.16– GUI for flying prescribed scenarios.

2.3.4.1 Select Profile & Open Models

To run a simulation, the user needs only to click one of the buttons in the SELECT PROFILE panel at the left of the GUI window and click the RUN THE SIMULATION button after the Simulink model appears. The final time has been set by the initialization code for the selected profile, so the value that appears in the final-time field of the model is ignored, unless the user starts the run manually.

The profiles available are:

- Burst, followed by a chop, at sea-level static, standard day conditions
- Chop, followed by a burst, at sea-level static, standard day conditions
- 150-sec takeoff run
- Climb from FC02 [sea level, Mach 0.25, TRA 100] to 40,000 ft, Mach 0.85, TRA 100, standard day
- Start at FC09 [42,000 ft, Mach 0.84, TRA 100], descend to 40,000 ft, Mach 0.70, TRA 60, hold briefly, and then descend to sea level, Mach 0.20, TRA 60
- Five steps down of 1° TRA from FC08 [35,000 ft, Mach 0.84,TRA 100, standard day]
- Five steps up of 1° TRA from FC07 [25,000 ft, Mach 0.62, TRA 60, standard day]

Note that “burst” refers to a step increase in TRA from ground idle to max power; “chop” refers to a step decrease in TRA from max power to ground idle.

The same Simulink model is used for all of the profiles and is shown in Figure 2.17. Note that the TRA input profile is contained in the subsystem ‘TRA >> PCNfRdmd.’ If the user chooses to run the simulation manually, the user may change the desired final time and run the simulation (bypassing the default settings in the GUI).

2.3.4.2 Plot Results

When the model has completed running, plots can be produced by using the six plot buttons.

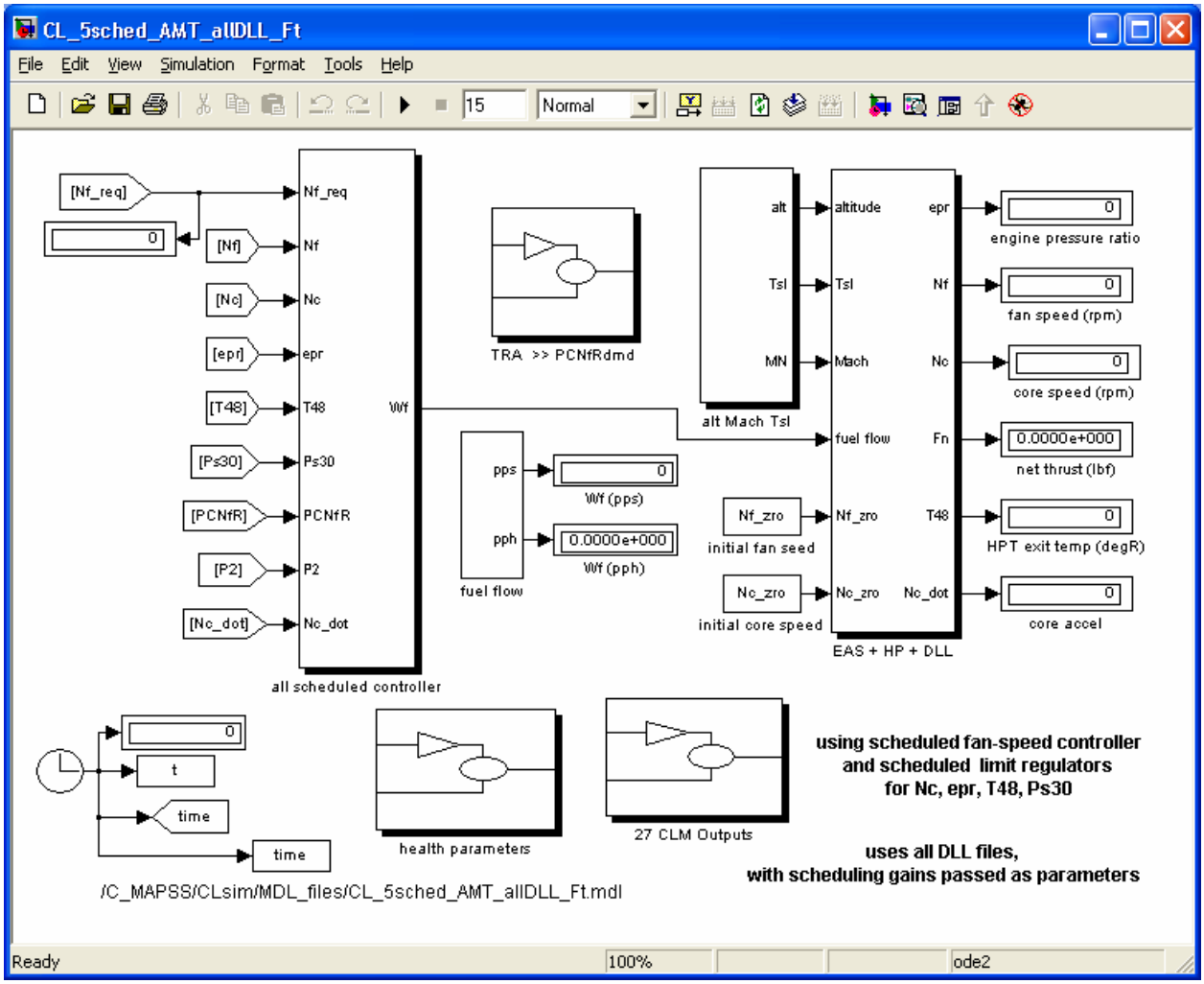

Figure 2.17.—Top-level of the model for flying prescribed scenarios.

3.0 Simulating the Response to Faults and Deterioration

Generally faults and deterioration in turbine engines are modeled by adjusting independent parameters associated with each component. These adjustments can also be used to represent engine-to-engine variation. They tend to shift the engine performance away from nominal. Deterioration is a slow process that occurs due to use; faults occur rapidly, although the resulting shift may be similar to that caused by deterioration. There are 13 health parameters in the 90K engine used to represent faults and/or deterioration. The values of the 13 health-parameter modifiers that are inputs to the fan, LPC, HPC, HPT, and LPT are determined by (i) the values of a set of six constant vectors that the user can define and (ii) the parameters of a step input that is part of the ‘health parameters’ subsystem. The six vectors are defined at startup in a manner that causes all of these inputs to be zero, regardless of how the step function is defined. Figure 3.1 shows the health-parameter subsystem available in each closed loop Simulink model (accessed by opening the ‘Health Parameters’ block at the top level of the model).

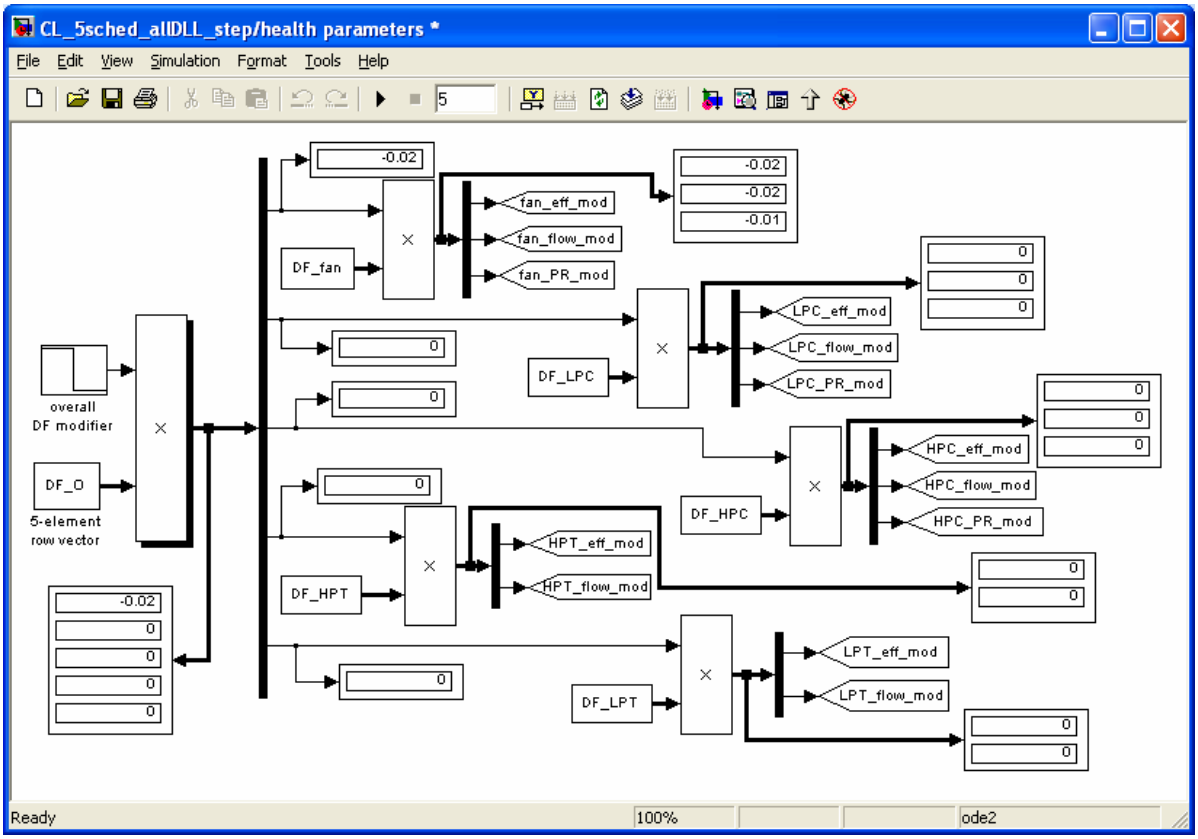

Figure 3.1.—Health-parameter subsystem.

Referring to the figure, we see that the multiplier at the left side has two inputs. The upper input is a step function, whose purpose is to establish the level of the fault or deterioration. Its parameters are initialized to start at zero and go to –0.02 at t = 1 sec, which corresponds to a 2 percent degradation. The lower input is the five-element row vector DF_O, whose purpose is to distribute the deterioration or fault to the five rotating components (fan, LPC, etc.). Its initial value is [0 0 0 0 0], which means that there is no deterioration and no fault in any of the components. As shown in the diagram, each of the five elements of DF_O multiplies another vector which distributes its value among the two or three modifiers for that particular component. For example, the vector DF_fan has three components that assign relative weightings to the modifiers of the fan's efficiency, flow, and pressure ratio. The DF_fan vector is initialized at [1 1 0.5], which means that if the default value is used, fan-efficiency and fan-flow modifiers will see all of the change produced by the product of the step function and the first element of DF_O, and the fan-pressure-ratio modifier will see half of this change. The same procedure is used for the other rotating components. Note that the two turbines have only two modifiers each, efficiency and flow.

3.1 Simulating a Fault

To simulate the response to an engine fault, the user must (i) set the final value of the step input of the health-parameter subsystem to have the desired final value, (ii) change the vector DF_O so that at least one of its elements is nonzero, and (iii) make any desired changes to the five individual vectors DF_fan, DF_LPC, etc. It is usually helpful to keep the TRA value constant during the run, so the demanded fan speed will remain constant, resulting in transient response due to the fault alone.

As a specific example, suppose we wish to simulate the response of the controlled engine to a sudden fault in the fan. To do this, we click the button labeled START AT POINT A on the top-level GUI shown in Figure 2.1, which brings up the GUI shown in Figure 2.8. Then we select one of the 14 flight conditions and click the third button on the left side of the GUI to build the closed-loop engine with the fully-scheduled fan-speed controller with limit regulators. At startup, the five-element row vector DF_O is initialized at [0 0 0 0 0], which means that the engine is in its nominal condition, without deterioration or faults.

To introduce a fault in only the fan such that at t = 1 sec the fan's efficiency, flow, and pressure-ratio modifiers take on values of –0.02, –0.02, and –0.01, respectively, we use the Matlab command window to set DF_O = [1 0 0 0 0]. Figure 3.1, which was obtained by double clicking the 'health parameter' subsystem on the top level of the model, shows the values of all 13 health modifiers at the conclusion of the 5-sec run. The step function input goes from zero to –0.02 at t = 1 sec and, because it multiplies the vector DF_O and because the fan-modifier vector DF_fan has been initialized at [1 1 0.5], the three fan modifiers change from their initial values of 0 to [–0.02 –0.02 –0.01]. Because only the first element in DF_O is nonzero, the 10 modifiers associated with the LPC, HPC, HPT, and LPT all remain at zero throughout the run.

In order to observe the effects of the fan fault, the TRA step input on the top-level model diagram is set to keep the TRA at its equilibrium value throughout the run by setting its final value equal to its initial value, namely `TRA_zro`. Hence, any transients that are observed will be due to the fan fault and the subsequent action of the fan-speed controller to maintain the fan speed at its demanded value. Figure 3.2 shows that the fan initially speeds up when the fault occurs and then returns to its original value as the fan-speed controller reduces the fuel flow. The 9-variable plot in Figure 3.3 shows the transients in fuel flow, epr, thrust, and other variables following the fault, all of which settle at lower values.

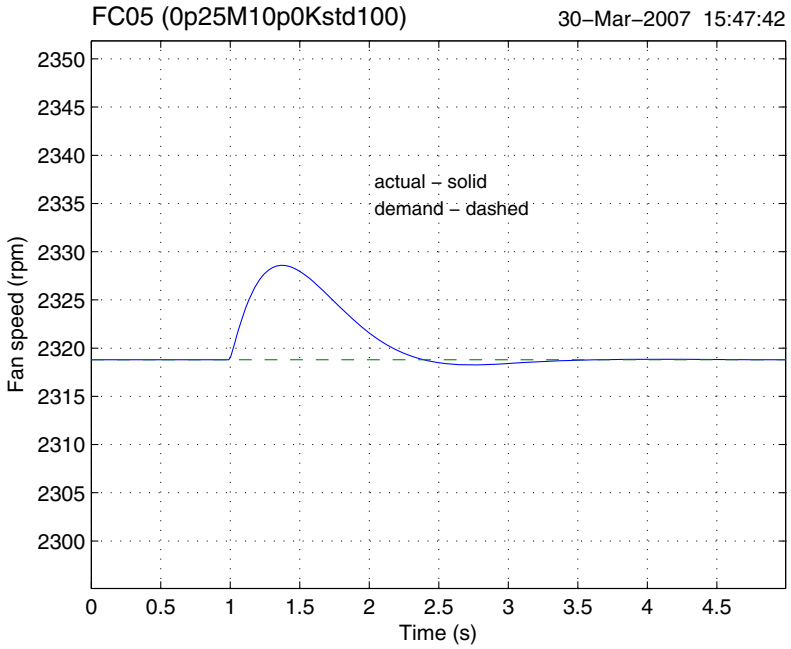

Figure 3.2.—Fan-speed response to a fan fault at constant TRA. Example generated at FC05.

3.2 Simulating Deterioration

In the health-parameter subsystem, we change the step function input to a ramp with a negative slope that simulates an overall deterioration level that increases with time (Figure 3.4). We will run the simulation for 40 sec, using a slope of –0.001, which will give a total deterioration level of –0.04, or a decrease of 4 percent in all the components (DF_O = [1 1 1 1 1]). Normally, the engine will be run to a constant fan speed, and the four limit regulators will be active. When the limit regulators are included in the control system, it is possible that the deterioration will cause one of the limit variables, such as T48, to exceed its design limit and thereby cause its limit regulator to determine the fuel flow, rather than the fan speed. For illustrative purposes only, because we are interested in simulating the long-term effects of deterioration regardless of the values of the variables, here we use the closed-loop engine model with *only* the fan-speed controller, and no limit regulators. Doing this, we obtain the response plot shown in Figure 3.5. We see that the fuel flow increases steadily, along with phi, and T48, while epr, thrust, and Ps30 decrease. Although the simulation was run over a 40-sec interval, we can translate time into deterioration level because a constant rate of deterioration (0.1 percent/sec) was used. Again, for this example run the fan-speed only controller was used, the limit regulators were not active.

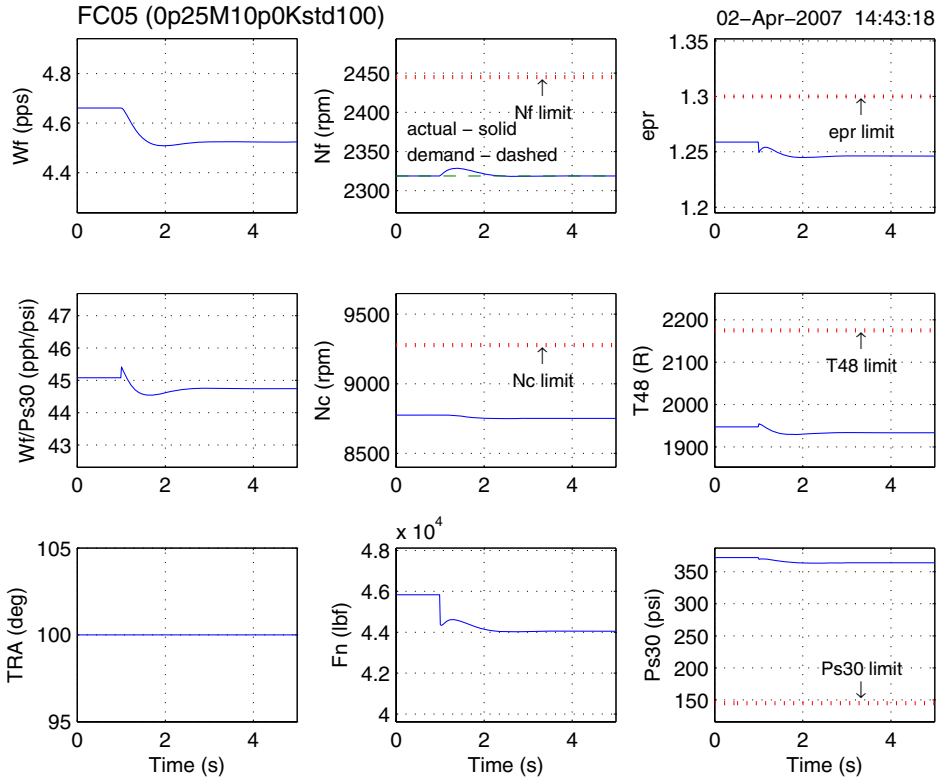

Figure 3.3.—Responses of key variables to a fan fault at constant TRA. Example generated at FC05.

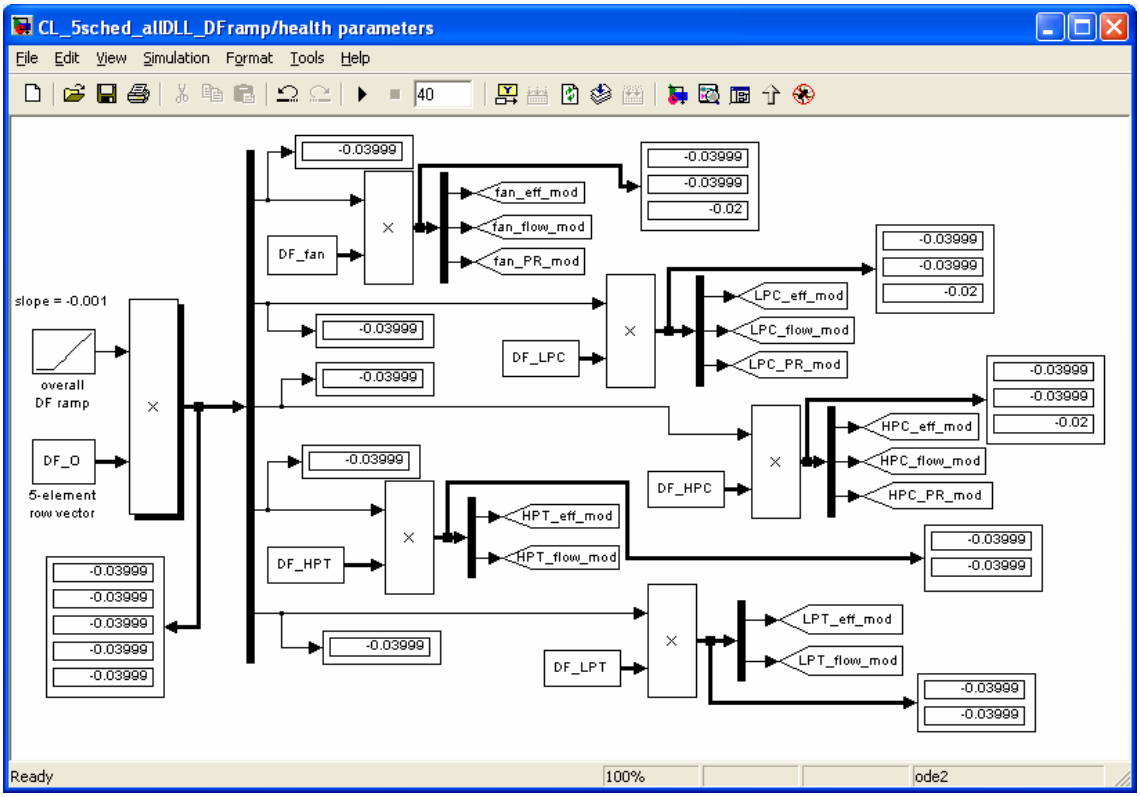

Figure 3.4.—Health-parameter subsystem with step input replaced by ramp input.

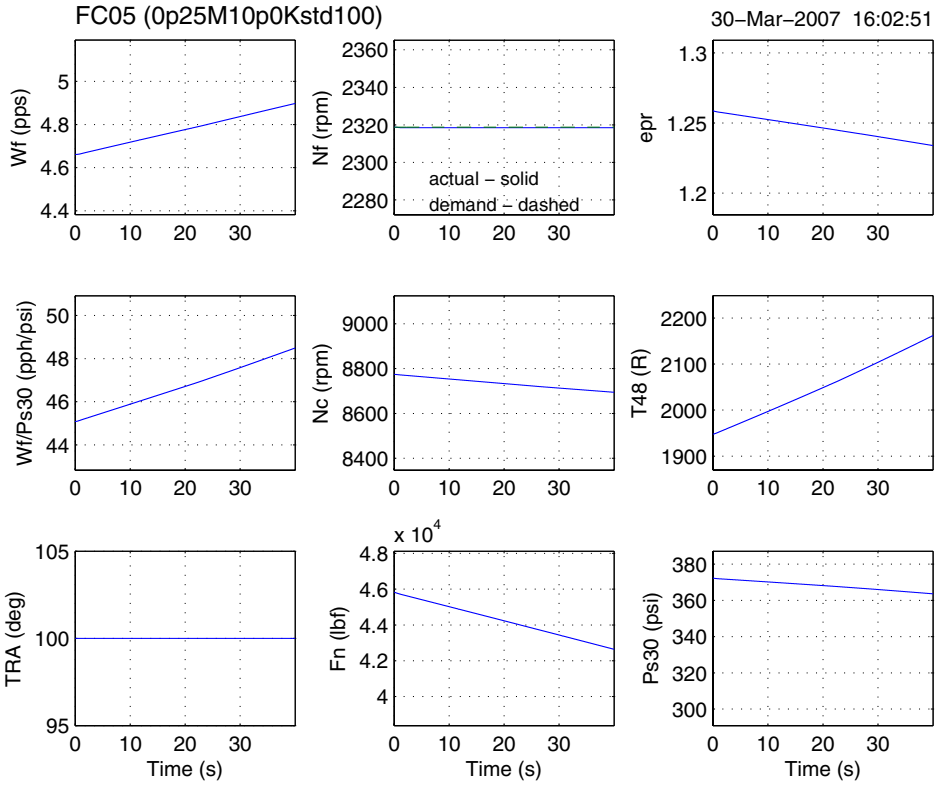

Figure 3.5.—Responses of key variables to deterioration in all five rotating components. Example generated at FC05.

4.0 Modifying C-MAPSS

In this section, we illustrate a variety of ways in which the user can extend the capabilities of C-MAPSS.

4.1 Using a Custom Controller

The control system can be modified by the user in two ways. First, the entire control system can be replaced with something that functions in an entirely different manner from the one that is provided with C-MAPSS, such as a model-predictive controller. Alternatively, the user might keep the same controller structure (fan-speed control with limit regulators), but choose a different design algorithm. The steps for doing both of these are outlined below.

4.1.1 Replace Entire Controller

The user may choose to replace the default controller with an imported controller block of their own. If this is the case, the user is instructed to remove the existing controller subsystem and save a closed-loop model file under a different name with the customized controller in place of the default one. Performing such a modification is simple and is illustrated by the following example. In this example, it is desired to alter the controller for the case where the engine is to fly with a prescribed trajectory as a function of time. First, the Simulink model `CL_5sched_AMT_DLL_Ft.mdl` is opened, which will appear as shown in Figure 2.17. This is the model file used when FLY PRESCRIBED FUNCTIONS OF TIME is selected in the TOP-LEVEL GUI. The user should then save the file under a different name (and if desired, in a new subfolder). At this point, the user can delete the block labeled 'all scheduled controller' and import the desired controller as a subsystem or Simulink block. In the configuration shown in Figure 4.1, only the fan speed demand and the measured fan speed are used by the controller; the other sensed outputs have been deleted. If it is desired to keep some of the sensor outputs for later use (e.g., incorporating limit logic), it is recommended to make use of the 'terminator' block, as shown in Figure 4.2.

In order to retain most of the GUI functionality, the user must perform the above tasks on several model files. Specifics for doing so are outlined in "Tips for modifying all Simulink model files" below. In some instances it may be more desirable to avoid having duplicates of the controller across several model files. This may be the case if changes to the controller are planned and the task of transcribing any change across several model files would be cumbersome and perhaps prone to error. Recommendations for this option are given in "Tips for modifying one Simulink model file" below.

4.1.1.1 Tips for Modifying All Simulink Model Files

If the user chooses to make use of the preconfigured flight scenarios, as described in Section 2.3, then the new controller must be incorporated into each model file that is called by the closed-loop GUIs. Table 4.1 summarizes the Simulink models that are associated with each closed-loop GUI. In this case, the user has access to all of the GUI features. The user should bear in mind that, when the BUILD MODEL or ACTIVATE MODEL buttons are used, the model file containing the custom controller must be opened manually. Table 4.2 shows the names of the files that set up the seven closed-loop flight scenarios.

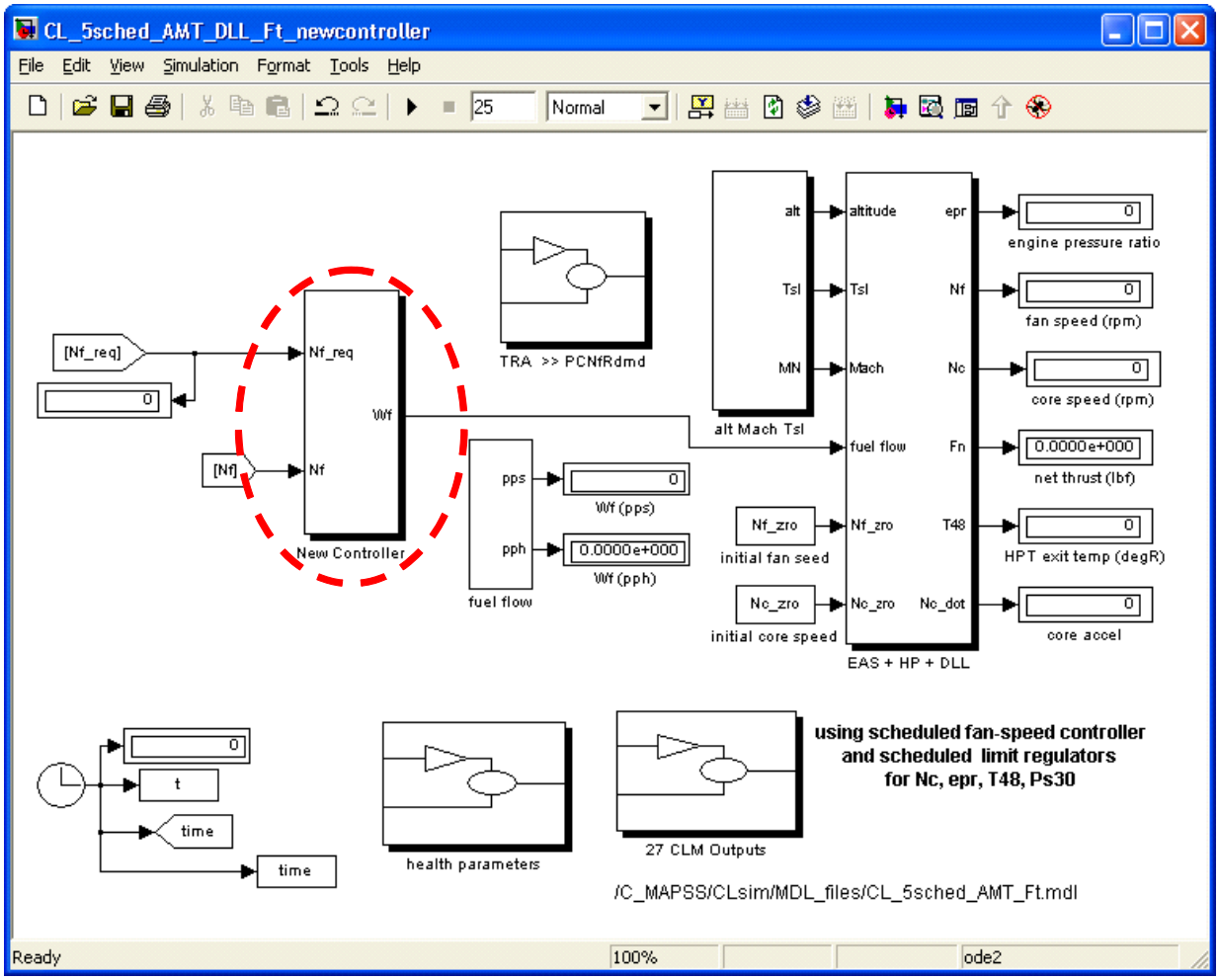

Figure 4.1.—Top-level Simulink diagram with custom controller created by the user.

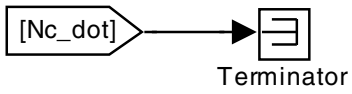

Figure 4.2.—Unused variable with a terminator block.

TABLE 4.1.—GUI AND MDL FILE NAMES FOR CLOSED-LOOP SIMULATIONS

Closed-loop scenario	GUI file	.MDL file
Start at Point A	`CLsys_14FCs_TRA`	(a) `CL_Nf_PG_TRA_DLL_step.mdl`
		(b) `CL_Nf_sched_allDLL_step.mdl`
		(c) `CL_5sched_allDLL_step.mdl`
Fly from Point A to Point B	`CLsys_fly_ABC_AMT`	`CL_5sched_AMTT_allDLL_SS.mdl`
Fly from A to B to C	`CLsys_fly_AtoB_AMT`	`CL_5sched_AMTT_allDLL_ABC.mdl`
Fly Prescribed Functions of Time	`CLsys_fly_FofT_AMT`	(d) `CL_5sched_AMTT_allDLL_Ft.mdl`

Notes:
(a) For use with unscheduled fan-speed controller (point gains).
(b) For use with scheduled fan-speed controller (no limit regulators).
(c) For use with fully-scheduled fan-speed controller and four limit regulators.
(d) Each flight scenario uses a different .M file in the `\c_mapss\CLsim` directory for setup of the inputs, as indicated in Table 4.2.

TABLE 4.2.—SETUP FILE NAMES FOR CLOSED-LOOP SCENARIOS.

Closed-loop scenario	Setup file
Burst & chop at SLS, std day	setup_BandC_run.m
Chop & burst at SLS, std day	setup_CandB_run.m
150-sec take off	setup_TO_run.m
Long climb	setup_long_climb.m
Long descent	setup_long_descent.m
Stair steps down	setup_steps_down.m
Stair steps up	setup_steps_up.m

4.1.1.2 Tips for Modifying One Simulink Model File

As an alternative to the recommendations above, the user may choose to alter only a single model file with the customized controller. If this is the case, the user would choose one of the models to import the controller and manually modify the engine inputs (TRA, altitude, Mach number, and ambient temperature) and manually load the desired flight conditions. In order to modify the input scenarios, the contents of the following blocks must be modified to vary the flight conditions and/or transient inputs: (1) the 'TRA >> PCNfRdmd' block, containing the TRA input transient and (2) the 'alt Mach Tsl' block, containing the environmental parameters. Note that doing this will circumvent any of the GUI functions. Because of this, the flight conditions may be defined by loading the appropriate .MAT file in the `\c_mapss\CLM` directory.

4.1.2 Developing Linear Controllers With Other Algorithms

All of the controller and limit-regulator designs that are part of C-MAPSS were done using the model-matching method of Edmunds (ref. 1). The point designs were done using the model-matching REGULATOR DESIGN GUI (Figure 2.5) and the scheduled controllers are based on designs that used the same algorithm, but were run in a batch mode. It turns out that this GUI can also be used to assist the user in creating linear controller designs with other algorithms, provided that the interface between the GUI and the rest of C-MAPSS is understood.

If the user wishes to create a linear fan-speed controller for one of the 14 flight conditions in the popup flight-condition menu, the REGULATOR DESIGN GUI can be used to help with the task. First, the flight condition should be selected. Then the other popup menu should be used to specify that it is the fan-speed controller that is being designed. Next, the user should click the button labeled BUILD DESIGN PLANT, which places the state-space LTI object `SSplant_siso` in the Matlab workspace. It is important to be aware that the free integrator, which ends up as part of the controller, is part of the design plant at this stage.

Now the user can carry out the design of whatever controller is desired, provided only that it produce the *scaled* incremental controller in the workspace with the name `SSreg_sc`. Once this has been done, two versions of the *unscaled* controller will be generated with the names `SSreg_unsc` and `SSreg_Nf`. At this point, the analysis buttons of the GUI can be used to display the incremental part of the controller in four different forms, and to draw a Bode plot of its magnitude and phase, versus frequency. Also, the closed-loop response of the fan speed and fuel flow to a step change in demanded fuel flow can be displayed. Finally, the controller can be saved in a binary file for use at a later time.

To help the user in taking advantage of this feature, there are three .M files in the folder `\Other_designs`. One of these (`design_code_template.m`) provides a template that the interested user can adapt for his/her own design algorithm. The other two files (`root_locus_design_example.m` and `gain_sweep_design_example.m`) illustrate how designs other than the model-matching method can be implemented. However, the user should understand that these two scripts do not represent complete design methods as they stand, because they both use the zero location of the model-matching design for the selected flight condition, rather than computing a zero location themselves. These files are intended only to demonstrate the process of matching the interface with C-MAPSS, so (i) the appropriate design plant can be produced with the proper name, and (ii) the

final design produced by the user's algorithm can reside in the workspace with the proper names so the analysis and save features of the GUI can be used.

4.2 Adding an Entry to the Flight-Condition Menu

Once the user has saved a flight condition as a binary .MAT file as described in Section 2.3.2, the data can be loaded from the FLIGHT CONDITIONS popup menu. By using the Matlab tool called GUIDE, which stands for 'Graphical User Interface Development Environment,' the GUIs may be modified to include these new flight conditions. Detailed help with GUIDE is available from the Matlab help facility. To use it, one needs to know that for each GUI, there is both a .M file for *running* the GUI, and a .FIG file for *developing* the GUI. As an illustration, we will go through the steps required to add the flight condition FC14 to the popup menu of the simulation CLSYS_14FCS_TRA GUI which is located in the `\c_mapss\CLsim` directory. For the purposes of this discussion, we assume that flight conditions FC01 through FC13 are already in the menu and FC14 is not.

To initiate the GUIDE tool, the user enters the command `guide` at the Matlab prompt, clicks on the tab labeled OPEN EXISTING GUI, and then clicks on the BROWSE button. The resulting window will show the .FIG files that exist in the directory to which it is pointing. Opening the file named `CLsys_14FCs_TRA.fig` produces the screen shown below in Figure 4.3.

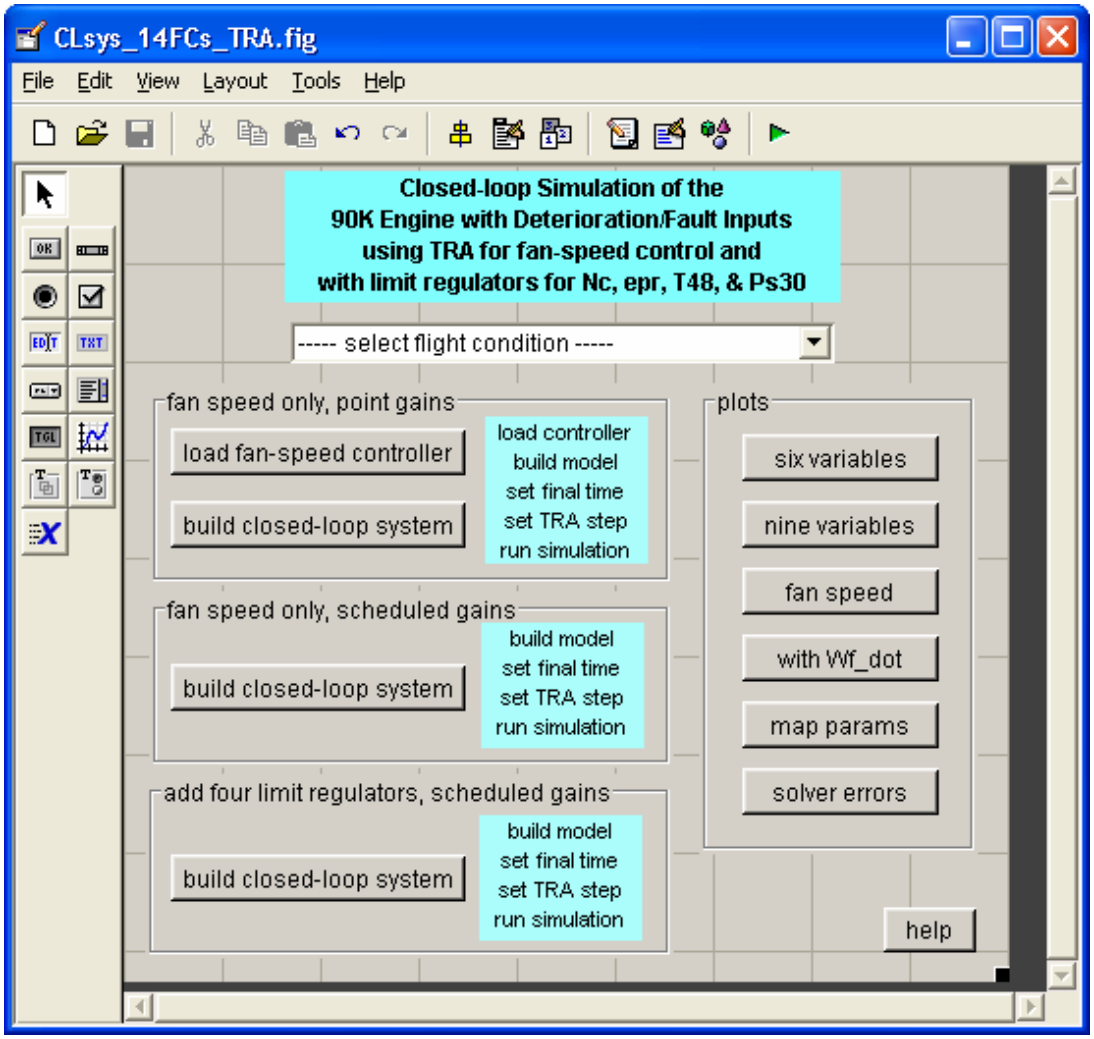

Figure 4.3.—Figure file for the GUI for starting at Point A.

Next, we double-click on the popup menu for the flight condition. Doing so will cause the Property Inspector window shown in Figure 4.4 to appear. Under the CALLBACK property is a Matlab script named `define_TRA_FCs_cbk.m` which contains the statements that are executed when the user clicks on one of the menu entries. Of interest here is the STRING property. If the user clicks on the string icon, the window shown in Figure 4.5 will appear. Figure 4.5 shows the portion of the string-property window starting with FC05, after the text for FC14 has been added at the bottom. When the new line has been added, the user clicks the OK button and the window closes.

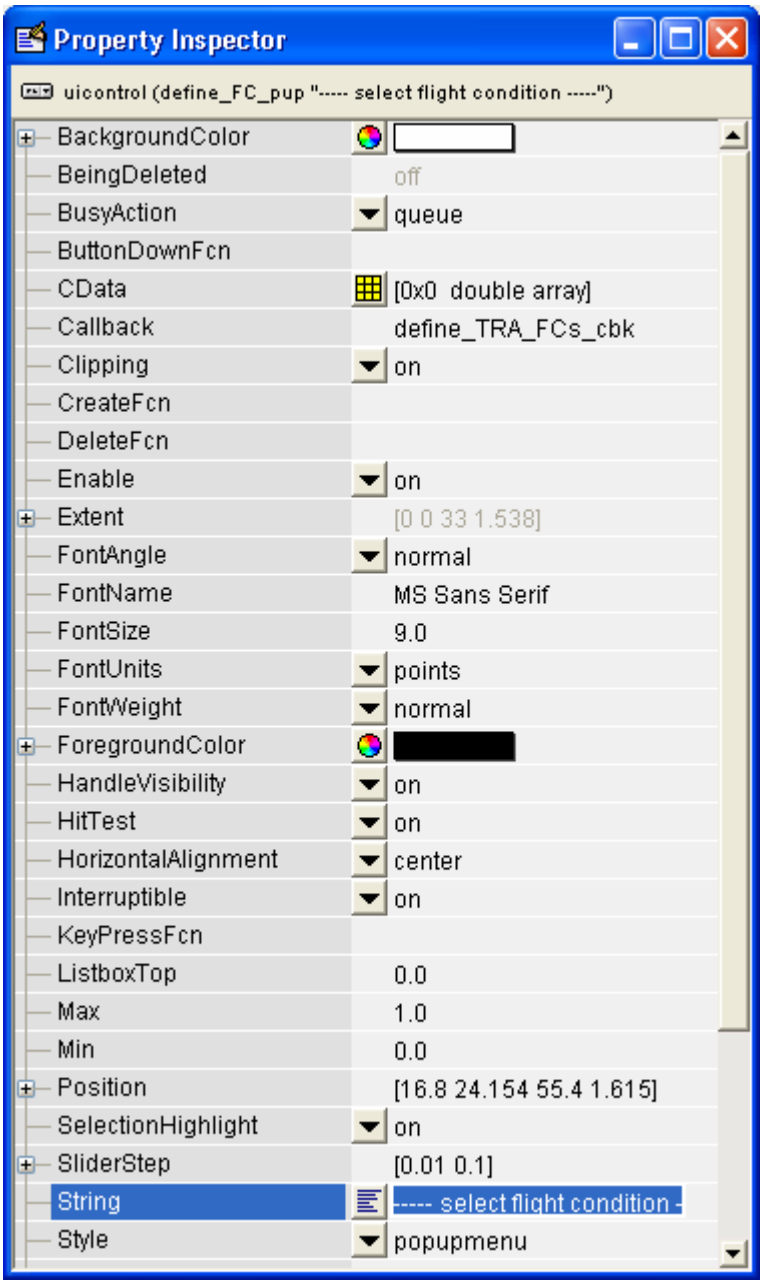

Figure 4.4.—Property-inspector window for the flight-condition popup menu.

The next task is to open the callback file `define_TRA_FCs_cbk.m`, which is in the `\c_mapss\CLM` directory, in the Matlab editor and add three lines of code that will handle the new flight condition. Figure 4.6 shows a portion of this file, as it appears in the Matlab editor. The text that has been added for FC14 consists of the last '`case`' statement of the '`switch`' command. When the callback executes, the variable '`str`' becomes the entire text that is the string property of the popup menu, and the variable '`val`' is set to the number of the line in the menu that has been clicked. Hence, when FC14 is clicked, `val` = 15 and the contents of line 15 in Figure 4.5 are compared with the string that follows the 15th '`case`' statement in Figure 4.6, namely, '`FC14: sea level,static,std day,TRA00`.' If the strings agree, then the '`load FC14`' command is executed and the message '`*** have loaded FC14.mat ****`' is displayed in the Matlab command window.

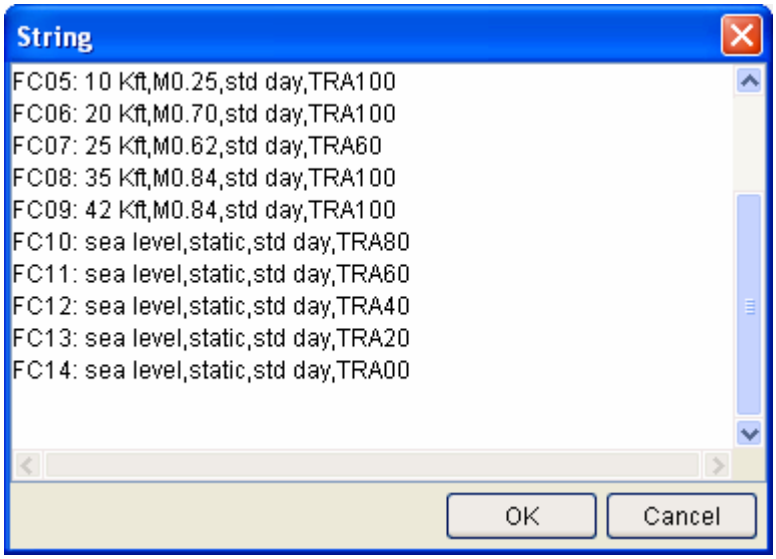

Figure 4.5.—String-property window for the flight-condition popup menu.

```
switch str{val};
  case '-- select FC --'
      disp('select flight condition...')
  case 'FC01: sea level,static,std day,TRA100'
    load FC01
    disp('*** have loaded FC01.mat ****')

:

  case 'FC13: sea level,static,std day,TRA20'
    load FC13
    disp('*** have loaded FC13.mat ****')
  case 'FC14: sea level,static,std day,TRA00'
    load FC14
    disp('*** have loaded FC14.mat ****')
  otherwise
    error('Illegal flight condition has been selected')
end
```

Figure 4.6.—Portion of code from flight-condition popup menu callback function.

Finally, we click on the TOOLS menu of the editor window and select RUN. This selection causes a new version of the GUI to be created that will have FC14 included as the last line of the popup menu used to select the flight condition. In order to include this new flight condition in other GUIs that have the popup menu for selecting the flight condition, we need to bring up the appropriate FIG file, use the property inspector to edit the string property, and then use the RUN command to create the .M file for the updated GUI.

4.3 Changing S-Functions

4.3.1 M-File S-Functions

In its normal form, C-MAPSS uses the compiled C-code versions (as DLL files in Windows) for the major engine components (fan, LPC, etc.). This has been done in order to maximize execution speed. However, this means that the user is unable to modify these S-functions without first modifying the source code and re-compiling. For convenience, the user can replace the DLL version of one or more of these components with the appropriate Matlab M file which can then be edited to make the desired modifications. Figure 4.7 shows the two versions of the fan S-functions and Table 4.3 gives the names of the M and DLL files for the S-functions of the major components. The M files can be found in the appropriate directories under the top-level C_MAPSS directory, such as `\Ambient`, `\Burner`, `\Fan`, etc.

TABLE 4.3.—NAMES OF THE M AND DLL FILES THAT IMPLEMENT THE MAJOR COMPONENTS OF THE 90K ENGINE.

Component	M file	DLL file
Ambient	`ambient_Mach_Tsl.m`	`ambient_Mach_C2.dll`
Inlet	`inlet_xxx.m`	`inlet_C.dll`
Fan	`fan_eff_W_PR_SM.m`	`fan_C2.dll`
LPC	`LPC_eff_W_PR_SM.m`	`LPC_C2.dll`
HPC	`HPC_bleed3_eff_W_PR_Ps30_SM.m`	`HPC_bleed3_C2.dll`
Burner	`burner.m`	`burner_C.dll`
HPT	`HPT_eff_W.m`	`HPT_C2.dll`
LPT	`LPT_eff_W.m`	`LPT_C2.dll`
Nozzle	`conic_nozzle.m`	`conic_nozzle_C.dll`
Schedules	`scheduled_gains_Nf.m`	`scheduled_gains_C2.dll`

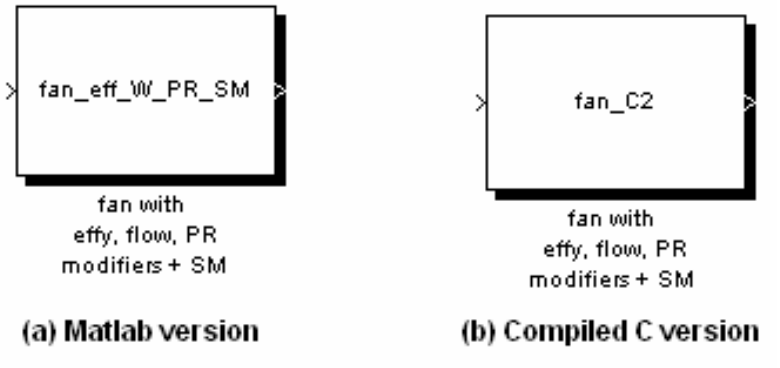

Figure 4.7.—Matlab and C versions of the fan S-function.

4.3.2 C S-Functions

Despite the ease of modifying Matlab-based S-functions, computational efficiency is reduced and real-time functionality is lost whenever M-files are used. On Linux and Mac OS, C-MAPSS cannot be run as delivered without first compiling the C-code into MEXGLX or MEXMAC files. Therefore, the original C source code has been included in the `\C source` directory along with the DLL files in the `\DLL` directory. The user may freely modify this code through the Matlab Editor. The code may then be compiled by typing (or copying and pasting) the script located at the header of the C-file at the Matlab command prompt. For example, in the file named `ambient_Mach_C2.c`, the header instructs the user to enter the following command:

```
>> mex ambient_Mach_C2.c interp2Ac.c
```

The `mex` command generates a platform-specific executable named `ambient_Mach_C2.<ext>` that calls the source files `ambient_Mach_C2.c` and `interp2Ac.c` (an interpolation function). Other components may be built using Table 4.3 as a reference (several ducts are implemented as S-functions, but these are not included in the table). Once the executable is created, it must be placed in the `\DLL` directory so that it is in the Matlab path. Note that Windows platforms running Matlab R14 SP3 or later will not build a DLL file, but instead a MEXW32 file. DLLs may still be used in an S-function, but if a given file name with both extensions resides in a directory, bear in mind that the S-function will default to the MEXW32 build.

In order to generate executables, it is of course necessary to have a C compiler installed on the machine. LCC is included in certain Matlab builds, otherwise freeware can be downloaded from the Web (e.g., GCC or Borland C/C++). In order to set up the appropriate compiler in Matlab, it is necessary to invoke `mex -setup` at the command line before compiling the code.

4.4 Adding Sensor and Actuator Dynamics

Recall that the Simulink models for the 90K engine use ideal sensors and actuators. As such, these sensors and actuators are not shown in the diagrams because they are just unity gains, with no dynamics. Should the user wish to include actuator and sensor dynamics, the Simulink diagrams can be modified as shown in Figure 4.8. In this example, we started with the Simulink model shown in Figure 2.9, which is for fan-speed control without limit regulators, and point gains. After simplifying the diagram by removing elements that are not involved with sensors and actuators, we have included a fan-speed sensor just before the lower input to the controller subsystem, and a fuel-metering valve (FMV) between the controller output and the fuel-flow input to the engine. Each of these new elements is a linear first-order system with unity low-frequency gain. The fan-speed sensor has a bandwidth of 1/0.02 = 50 rps, and the FMV has a bandwidth of 1/0.05 = 20 rps. Should the user desire to include sensor noise or bias in the model, it is a simple matter to insert a summing junction at the output of the sensor. One should keep in mind that the controller gains have been determined based on the assumption that there are no sensor or actuator dynamics present. Hence, the controllers that come with the 90K model will not work as well when sensor and/or actuator dynamics are present. In order to obtain the same quality of response, it is necessary to redo the point gains with the sensor and actuator models included in the design plant. Furthermore, none of the scheduled gains should be expected to work as well with sensors and actuators included, because the scheduling tables are based on designs done without sensors and actuators.

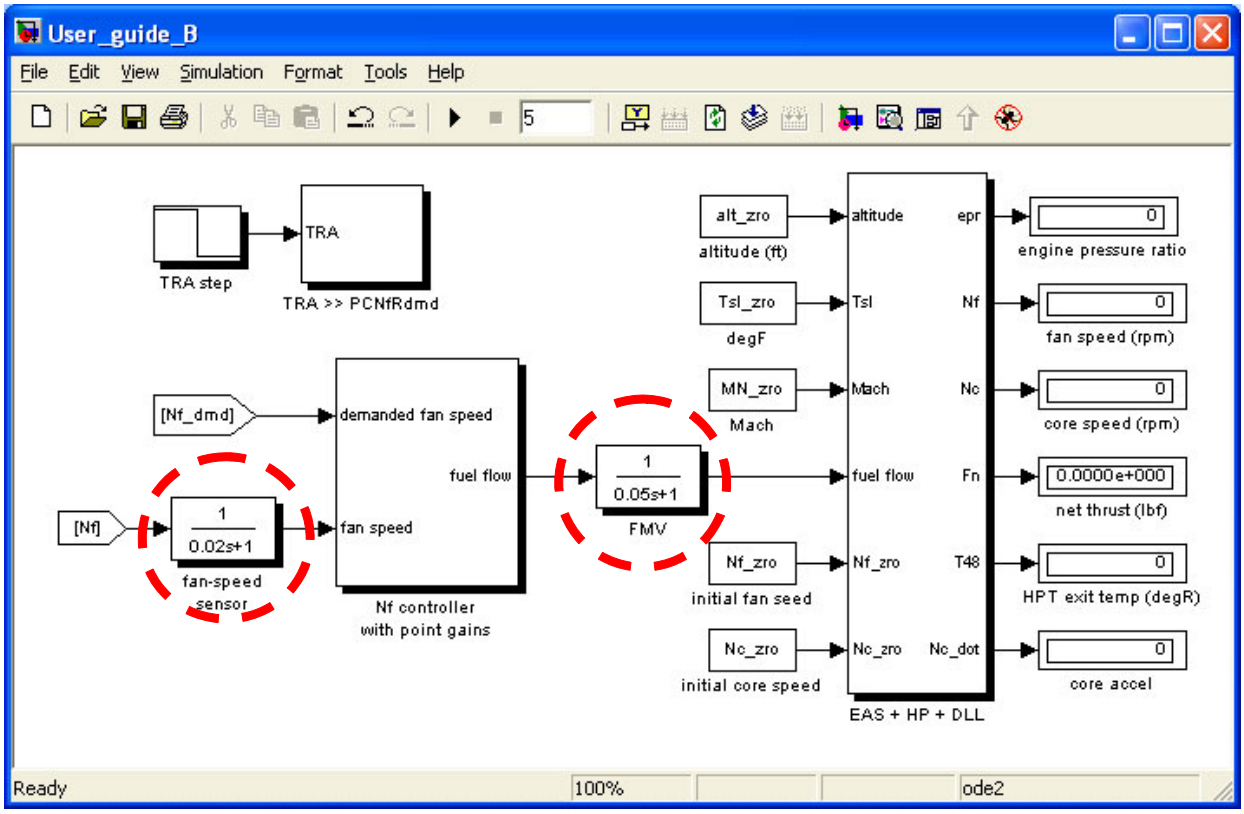

Figure 4.8.—Top-level Simulink diagram with fan-speed sensor and fuel-metering valve (FMV) added.

4.5 Adding an Input or Output to an S-Function

Should the user need to have an additional input or output in one or more of the S-functions that come with C-MAPSS, this can be done by following the steps outlined below. In the current version of C-MAPSS, the health-parameter modifier inputs to the fan, LPC, HPC, HPT, and LPT were added after their original S-functions had been created. Also, the stall-margin outputs of the fan, LPC, and HPC S-functions were added later.

Because the DLL files contain compiled C code, the Matlab version of the S-function is included in order to easily make certain changes. For example, from Figure 4.7 we can see that the file for the Matlab version of the fan is `fan_eff_W_PR_SM.m`, which resides in the directory `/c_mapss/fan`. In the following sections, we will show how these additions were made.

4.5.1 Adding an Input

Near the beginning of the `fan_eff_W_PR_SM.m` file, in the section labeled `case 0`, there is the line

```
sizes.NumInputs = 7; % added three for health params
```

When the three health-parameter modifier inputs were added, the value of `sizes.NumInputs` was increased from 4 to 7. A few lines further down, in the section labeled `case 3`, there are the following three lines of code:

```
effMod = u(5);   % effy modifier
Wmod   = u(6);   % flow modifier
PRmod  = u(7);   % PR modifier
```

These instructions (i) establish the ordering of the three new inputs, namely efficiency, flow, and pressure ratio, (ii) place them after the existing four inputs, and (iii) assign meaningful names (`effMod`, `Wmod`, and `PRmod`) for their use in the calculations. When the name of the modified S-function is entered into the S-function block in the Simulink diagram, the three new inputs will appear on the left side of the block, as shown in Figure 4.9. At this point, all that remains is for the user to make the appropriate connections in the diagram.

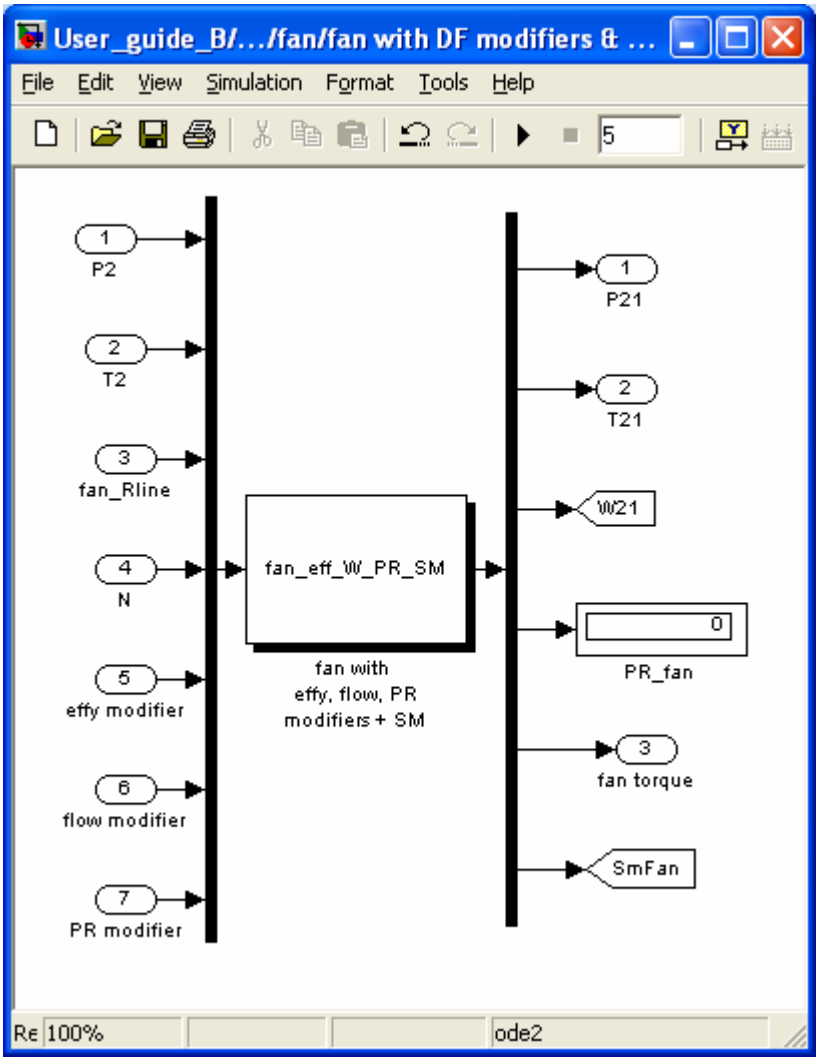

Figure 4.9.—Simulink diagram of the Matlab version of the fan S-function, with the three modifier inputs and stall-margin output.

4.5.2 Adding an Output

Also in the section labeled `case 0`, there is the line

```
sizes.NumOutputs = 6;   % added one for stall margin
```

When the stall-margin output was added, the value of `sizes.NumOutputs` was increased from 5 to 6. Also, code was added to the body of the S-function to compute the stall margin as the variable `SM`. At the end of the section labeled `case 3`, the following line was added in order to have the stall margin become the sixth output:

```
sys(6) = SM;   % constant-flow stall margin
```

In the lower right part of Figure 4.9, we see that the stall margin appears as the sixth output. The same approach can be used to add any variable to the output of the model, provided that the variable exists internally or the code to compute it can be incorporated.

5.0 Steps Used to Develop Scheduled Fan-Speed Gains for the C-MAPSS Engine Controller

In order to be able to do the large number of controller designs required for the gain scheduling without having the user do the point-and-click operations required by the design GUI, a set of Matlab files was created in the directory `/BATCH_files`. These files are organized in the two sub directories: `/CLM` and `/Design` and are separate from any of the GUI-based code.

Step 1: The flight-condition (FC) files and the linear engine model (LEM) files were developed using the code in the `/CLM` path. The LEM files were then copied to the `/Design` path where they were used to produce designs, from which the scheduling tables were developed.

Step 2: In `/CLM`, the Matlab script `make_AMTT_EE_FCs.m` was run to produce 15 FC files for the user-specified TRA value, where the user is prompted to enter `[100|80|60|40|20].` This produced 15 binary files named `EExx.mat` in the `/CLM` directory. These files were then moved manually to the directory `/CLM/EE_FC_files` for use in developing the LEMs. Note: During the development process, flight conditions were assigned an index number in the range [10,84] in order to have two digits in the index. The scheduling CLMs used here have one input (Wf) and only five outputs (Nf, Nc, epr, T48, and Ps30). They are *not* the same as the CLMs used for simulation, which have 14 inputs and 27 outputs. Also, note that all of the flight conditions used for the scheduling calculations are for standard-day temperature (59 °F).

Step 3: The GUI named `CLM_1x5_batch.m` was used to do the following:

(a) Load one of the FC files `EExx.mat`,
(b) Perform a 5-sec equilibrium run to verify that the engine started out in equilibrium, and
(c) Create the LEM and save it as a binary file. It was helpful to see the value of x1_dot (the time derivative of the state variable x1) displayed for the base case (it should be close to zero) and for the positive and negative perturbations in the two state variables (x1 and x2) and the input (Wf). The positive and negative values should have approximately the same magnitudes and opposite signs. It was also helpful to look at the two poles of the LEM (both should be negative). The zeros should be `NaN`. The LEM files were named `LEM_EExx.mat` and ended up in the directory `/CLM/LEM_files/EE_series`. There were 75 of them, where '`xx`' goes from 10 to 84. The

files were also copied to the directory `/Design/LEM_files/EE_series` for use in the controller-design process.

Step 4:

(a) The GUI named `/Design/DESIGN_1x5_batch.m` was used to do a few designs for the fan-speed controller in order to check things out,

(b) The batch file `/Design/design_Nf_regs_batch_EE.m` was used to do all 75 designs and to summarize the time- and frequency-domain specifications (percent overshoot, rise time, peak time, settling time, σ_{max}, bandwidth based on -3dB magnitude or 60° phase lag, and phase margin).

(c) the Matlab script `/Design/show_CL_perf.m` was used to generate plots of fan speed (Nf) and fuel flow (Wf) for a step in demanded fan speed, using the linear model.

Step 5: The Matlab script `/Design/calc_CL_perf.m` was used to make tables of the time- and frequency-domain specifications.

Step 6: The script `/Design/create_Nf_ZZZ_array.m` was used to produce the 75x7 array `ZZZ` that contains `[ii a b c d PCNfR P2]` for all of the designs, where `ii` is the index value, `PCNfR` is the percent corrected fan speed, `P2` in the fan inlet pressure, and the quantities `a, b, c`, and `d` are the state-space matrices of the incremental part of the fan-speed controller. This array was saved in the binary file `/Design/Schedule/fan_speed/Nf_reg_vals_EE.mat`. It may turn out that some of the flight conditions do not produce reasonable LEMs (this is most likely to occur at low Mach number, high altitude, and low TRA values). If this happens, these designs can be removed as part of the next step.

Step 7: The script /Design/Schedule/fan_speed/make_surface_data_nf.m was used to do the following:

(a) Remove any flight conditions for which the design is not to be used to compute the scheduling tables (e.g., those that did not produce adequate LEMs). In this case, two flight conditions were removed, namely for `ii` = 37 [40 Kft, Mach 0.5, TRA80] and for `ii` = 49 [30 Kft, Mach 0.40, TRA60], leaving 73 design for scheduling. The B and D gains can be plotted versus the index `ii` in order to help spot outliers that should be removed.

(b) Check that the values of the C gain are all the same, as only the B and D gains are to be scheduled. For all of these designs in the EE series, C turns out to be 32. However, for some of the limit-regulator designs, C turns out to be 16 rather than 32 (there must be a choice made somewhere in the model-matching design code). It can be shown that if the product BxC remains the same, the controller transfer is not affected. Hence, we can change C from 16 to 32 merely by dividing B by 2, thereby keeping the product the same.

(c) After any rows corresponding to bad designs have been removed from the array `ZZZ` (two, in this case), the new array was designated as `YYY` and was saved in the binary file `surface_data_Nf_EE.mat`. Also saved in this file were:

(i) `P2`, the vector of 21 fan-inlet pressure values used for the scheduling tables,

(ii) `NF`, the vector of 19 PCNfR values used for the scheduling tables,

(iii) `BB`, the 21x19 array of B gains that will be used to compute the scheduled B gain, and

(iv) `DD`, the 21x19 array of D gains that will be used to compute the scheduled D gain.

The 19x21 = 399 values of `BB` were determined by using the ‘nearest neighbor’ approach, where the four designed gains closest to a mesh point were used. To make this approach work, it is necessary to

assign a scale factor so deltas in P2 can be equated to deltas in PCNfR. This was done by computing the ratio of ($PCNfR_{max}$ – $PCNfR_{min}$) to ($P2_{max}$ – $P2_{min}$). Because the values of PCNfR range from 50 to 100 percent and the values of P2 range from 3 to 18 psia, the scale factor is (100–50)/(18–3) = 50/15 = 3.333. Once the scale factor had been determined, the contribution of each of the four neighboring design gains was weighted inversely according to the distance from the particular mesh point. The two scheduling tables (BB and DD) were saved in the binary file Nf_schedule_arrays.mat, along with the vectors NF and P2 that define the mesh points.

Step 8: To assist the user in visualizing the gain schedules and in detecting inconsistencies or outliers, the script plot_gains_Nf_reg.m was used to produce a 3–D plot of the designed gains, versus PCNfR and P2. No scheduling information was included.

Step 9: Three-dimensional plots of the final surface (versus the two scheduling variables PCNfR and P2) that defines the scheduled gains are produced by the script plot_surfs_Nf.m, where color is used to represent the magnitude of the gain. The design gains are superposed on the plot as black dots. By interactively changing the viewing angle of the 3–D surface, the designer can get an excellent visualization of the closeness of the fit.

Reference

1. Edmunds, J.M., "Control System Design and Analysis Using Closed-Loop Nyquist and Bode Arrays," International Journal of Control, vol. 30, no. 5, 1979, pp. 773–802.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-10-2007	Technical Memorandum	

4. TITLE AND SUBTITLE
User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Frederick, Dean, K.; DeCastro, Jonathan, A.; Litt, Jonathan, S.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
WBS 457280.02.07.03.04.03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135-3191

8. PERFORMING ORGANIZATION REPORT NUMBER
E-16205

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001
and
U.S. Army Research Laboratory
Adelphi, Maryland 20783-1145

10. SPONSORING/MONITORS ACRONYM(S)
NASA, ARL

11. SPONSORING/MONITORING REPORT NUMBER
NASA/TM-2007-215026

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Categories: 07, 09, and 63
Available electronically at http://gltrs.grc.nasa.gov
This publication is available from the NASA Center for AeroSpace Information, 301-621-0390

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report is a Users' Guide for the NASA-developed Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) software, which is a transient simulation of a large commercial turbofan engine (up to 90,000-lb thrust) with a realistic engine control system. The software supports easy access to health, control, and engine parameters through a graphical user interface (GUI). C-MAPSS provides the user with a graphical turbofan engine simulation environment in which advanced algorithms can be implemented and tested. C-MAPSS can run user-specified transient simulations, and it can generate state-space linear models of the nonlinear engine model at an operating point. The code has a number of GUI screens that allow point-and-click operation, and have editable fields for user-specified input. The software includes an atmospheric model which allows simulation of engine operation at altitudes from sea level to 40,000 ft, Mach numbers from 0 to 0.90, and ambient temperatures from -60 to 103 °F. The package also includes a power-management system that allows the engine to be operated over a wide range of thrust levels throughout the full range of flight conditions.

15. SUBJECT TERMS
Commercial turbofan engine simulation; FADEC; Propulsion control; Digital simulation; Dynamic system; Models; Computer aided design; Control system design; Engine control

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email:help@sti.nasa.gov)
U	U	U	UU	45	19b. TELEPHONE NUMBER *(include area code)* 301-621-0390