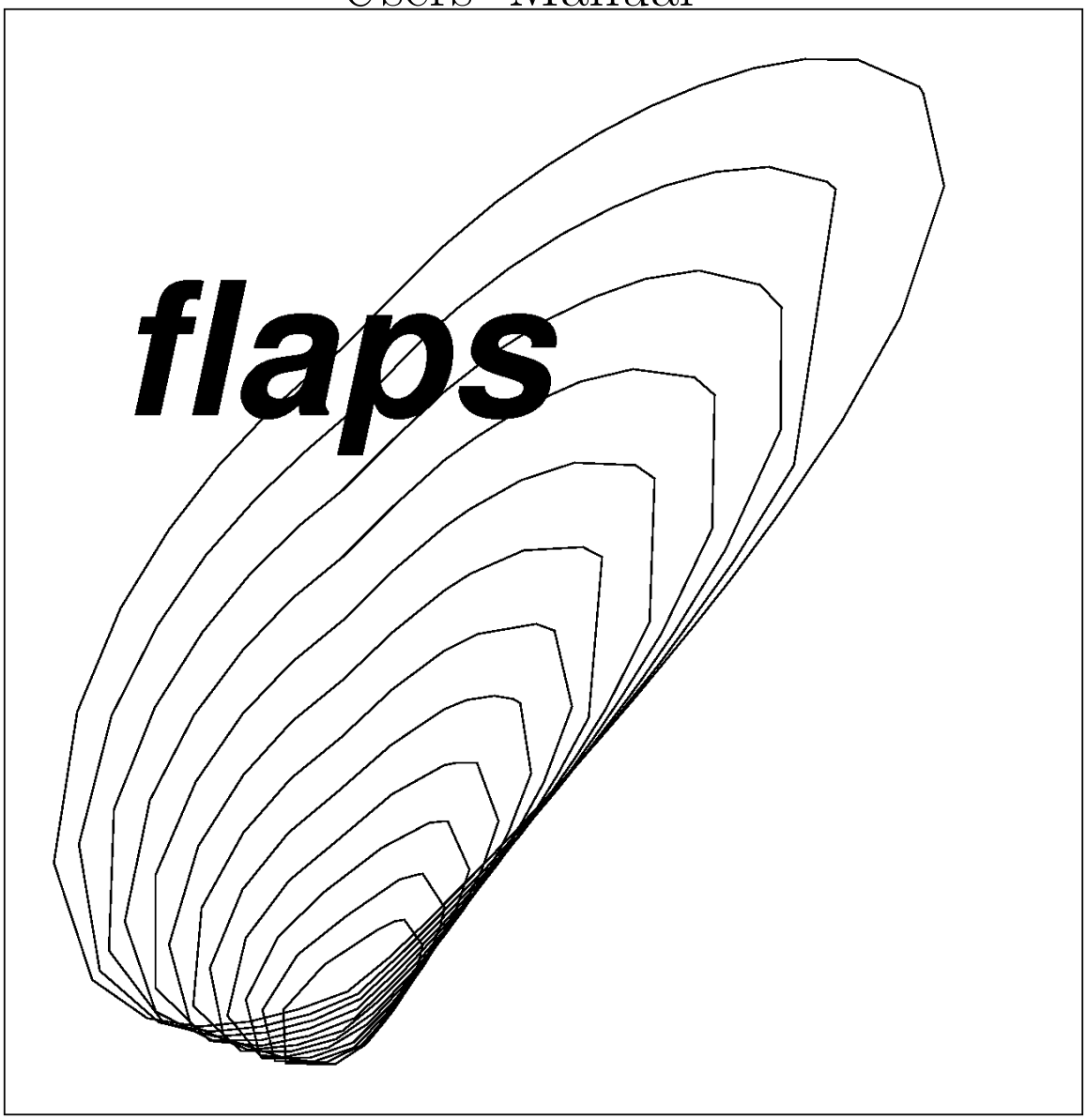
Users' Manual



Ed Meyer
Flutter Methods Development

June 9, 2018

Part I

Theory

Contents

Chapter 1

Introduction

This manual describes FLAPS, a collection of programs that work together through a common database to perform various structural dynamics analyses in each of the three basic categories of structural dynamics: vibration, stability, and response. It is intended both as a reference and a guide to the theory underlying FLAPS capabilities.

Most engineering analyses can be done in more than way, so one of the design goals of FLAPS is to facilitate different types of analyses. To this end FLAPS includes many ways to manipulate equations of motion, using commands like `alge`, `extract`, `merge`, `modes`, and `param`, or options within the solution commands `stab`, `fresp`, `tresp`.

Structural dynamics analyses often involve studying the effects of various parameters on the solution, whether to gain insight into the behavior of the structure or to analyze behavior at a range of conditions. For example, varying nacelle stiffnesses can give insight into the flutter characteristics of an aircraft, while varying fuel loading, altitude, and velocity give the flutter characteristics over a range of flight conditions necessary to certify an aircraft. Parameters play an important role throughout FLAPS. A number of parameters are pre-defined, along with equations which define them in terms of other parameters. New parameters may be defined and given arbitrary equations in terms of other parameters. Pre-defined parameters may be given new equations. The goal is to allow continuous parameter variations using the solution commands by parameterizing the matrices which make up the equations of motion.

Vibration problems and general polynomial eigenvalue problems are treated with the FLAPS `eigen` command. Dynamic response problems can be classified as time-domain, treated by the `tresp` command, or frequency-domain, treated by the `fresp` command. Dynamic stability analyses can be classified as flutter, treated by the `stab` command, and shimmy, treated by the `shimmy` command. Nonlinearities may be accounted for using the describing-function technique for frequency-domain analyses, or by directly specifying equations for matrix elements in the time-domain.

FLAPS has no model-building capability; structural models must be built in ATLAS, Elfini, NASTRAN or the Boeing Aeroelastic Process (BAP), [1]

[1] BAP is a special version of NASTRAN produced by the MSC.Software Corporation for Boeing.

then imported into FLAPS. The FLAPS import command reads data produced by these programs and more.

FLAPS analyses are *portable* between platforms in the sense that an FLAPS database can be saved (to an FLAPS savefile, possibly on a remote platform) and restored with the restore command (§10.19) on any other platform. This allows an analysis to be broken into phases with each phase running on the most appropriate platform. For example, an analysis might be run on the Cray to generate a ATLAS model and an FLAPS savefile with all the matrices necessary to perform subsequent flutter analyses. Then the savefile might be restored in an FLAPS job on a Linux PC cluster to do flutter analyses, saving the results on another savefile which might be restored in another FLAPS job on an RS/6000 to visualize flutter modes. It is an unfortunate fact of life that we cannot yet do all aspects of our analyses on a single platform; the portability feature of FLAPS savefiles is intended as a convenience until we can.

Structural dynamics analyses often comprise numerous commands with many arguments; writing FLAPS programs is not unlike writing code in a programming language such as Fortran or C++. As with these languages, writing FLAPS programs is not conducive to graphical user interfaces; a simple text editor is the tool of choice for creating FLAPS programs, as it is with other structural dynamics packages such as NASTRAN, ATLAS, and Elfini [47]. There is no way to avoid the fact that structural dynamics analyses are complex. The simplest way to run FLAPS is to create a file of commands, then type `apex` *`input-file`*; the primary purpose of this manual is to detail how to create the *`input-file`*.

Currently FLAPS is available on Unix (or its close cousin Linux) platforms only, including

- Boeing Linux PC Clusters (http://cluster.web.boeing.com)
- Desktop PCs with Linux (http://linux.web.boeing.com)
- IBM AIX SPs (http://aixgrp.web.boeing.com)

Following this introductory chapter are chapters giving details of the capabilities in various FLAPS modules and some of the algorithms used in them.

Chapter 2 how to run FLAPS and a quick start guide

Chapter 3 is a summary of the equations used at Boeing for stability, response, and vibration analyses.

Chapter 4 shows how matrices can be modified to change the equations of motion for use in stability, response, and vibration analyses.

Chapter 5 discusses parameters: using and modifying pre-defined parameters and defining new ones.

Chapter 6 describes how matrices can be made functions of parameters so that they may be used for parameter studies.

Chapters 7-10 detail the FLAPS capabilities for vibration, stability, and response.

Chapter 11 discusses the visualization capabilities in FLAPS

Chapter 12 lists the demonstration problems available.

Chapter 13 is a reference for all FLAPS commands.

1.1 Flaps Capabilities

The application programs comprising FLAPS fall into the following general categories:

- importing data from other programs
- data manipulation
- parameterization
- vibration, stability and response calculations
- data visualization

1.1.1 Importing Data

FLAPS has no structural-model-building capabilities, so it is usually necessary to obtain structural models from other sources. There are two ways of doing this in FLAPS, using files exported from other programs (several formats) and the `import` command (§10.12), or FLAPS binary savefiles and the `restore` command (§10.19). A number of formats are supported by the `import` command:

- NASTRAN or BAP database files (binary)
- `OUTPUT4` files created in NASTRAN or BAP (ASCII)
- `dmig` file created in NASTRAN or BAP (ASCII)
- `neutral` files created in Elfini `feintr` (base module) or `extraction` (aeroelasticity module) (ASCII)
- Matlab or Simulink `MAT` files (binary)
- Matrix Market - a simple format well suited for large and/or sparse matrices, popular in the numerical analysis community [24]. (ASCII) Used by the FLAPS print command to display large matrices.
- `ESA` plot files created by a variety Boeing programs, including FLAPS. (ASCII)

Of these, the ASCII files are editable and portable across platforms, unlike the binary files which are neither. Binary files are generally much smaller than the equivalent ASCII file.

The other way data may be introduced into FLAPS is by creating an FLAPS savefile which can be read with the `restore` command. FLAPS savefiles are binary, so you cannot edit them, however they use a special format that makes them portable across computer architectures, and they are much smaller than equivalent ASCII files. FLAPS savefiles may be created using

- the FLAPS `save` command (§10.20)
- the `apex import` command with a NASTRAN or BAP database (10.12.3)
- the `apex extract` command with an Elfini database (A.1)
- the ATLAS `SAVE` command (A.2)

1.1.2 Data Manipulation

Matrix algebra (addition, subtraction, relative difference, transposing, inversion, pseudo-inverse, and scaling) is available in the `alge` module for all FLAPS matrices. Parameterized matrices may be manipulated at specified values of their parameters.

Slices of rows and/or columns may be extracted from FLAPS matrices using the `extract` command (§10.9). If nodal data are available the rows and columns may be specified using node/freedom numbers.

Matrices may be merged to form new matrices using the `merge` (§10.13) command.

Rigid-body modes may be created and used to replace existing rigid-body modes using the `modes` command. The `modes` command also has the ability to change coordinate systems associated with a modes matrix.

1.1.3 Parameterization

An important aspect of FLAPS is its ability to do parameter studies. The key to parameter studies is the ability to define parameters and to create matrices which are functions of these parameters. Several types of matrix parameterization are available in the `param` module (chapter 6), including multi-variable interpolation, rational-function approximation, branch-mode frequencies, structural-damping, replacing matrix elements with arbitrary equations, ABCD control-laws, and the most general: user-written subroutine parameterization.

1.1.4 Vibration, Stability, and Response

The FLAPS `eigen` command solves a number of types of eigenvalue problems including vibration and buckling. The `stab` command solves linear flutter equations in the fre-

quency domain for neutral stability and parameter variations. Nonlinearities may be included using the describing-function technique (§**??**). Dynamic response problems in the frequency domain are solved in `fresp`, or in the time domain with the `tresp` command.

1.1.5 Data Visualization

Matrices, unless they are very small, are difficult to examine on the printed page so FLAPS displays them graphically in a way that allows interactive visualization. The print command (§10.16) is the primary way to visualize matrices.

Results from the computational modules can be plotted in 2D using the vis command (§10.21); if nodal and modal data are available, animated modes can be visualized by selecting points from a `vis` plot.

ESA-formatted plot files [15] are created by `param`, `stab`, `fresp`, and `tresp` which may be processed in the Boeing program `pegasus` [34], good for documentation, but not very good for quick looks. The FLAPS `vis` command may also be used on the command-line with any ESA-formatted file; see the `vis` reference (§10.21.5) for more details.

1.2 Typographic Conventions

Throughout this document the following typographic conventions are used to (hopefully) make it more readable:

glossary is used to highlight new terms and concepts when they are first used

`Typewriter` is used for computer output or listing of computer files

`Italic Typewriter` indicates a parameter which the user should substitute with an actual parameter; for example

flaps *`filename`*

means that the user is to substitute her own file name for *`filename`*

San Serif is used for a computer command or the name of a program, for example flaps

Bold San Serif indicates something typed by the user; for example

`$Login:`**eem2314**

$ is the traditional Unix command prompt (yours will probably look different).

Ctrl starts a control character. For example, to type Ctrl-d hold down the control key (sometimes marked Ctrl or Ctrl/Act) and press the "d" key.

M Matrices and vectors are printed in bold, with matrices in upper case and vectors in lower case.

1.3 Getting Started

Throughout this document we assume the reader has a rudimentary knowledge of the Unix operating system, and is familiar with terms like environment variable, directories, and can use an editor such as vi or emacs. For more details on these and other concepts see [29], [16], and [1].

1.3.1 The flaps Command

The basic program used in running FLAPS is named flaps (actually it's a script which you can edit), which should be in your default `PATH`. While the most common use of flaps is for processing files of FLAPS commands, flaps has other options (see §10.6 for more details):

`apex amvis` *`file`* visualize modes contained in a Universal-formatted file

`apex clean` remove any FLAPS temporary directories left from failed runs

`apex demo` change the current working directory to where the FLAPS demonstration files (chapter 9) are kept.

`apex dog` works like the Unix command `cat` except that environment variables are replaced.

`apex help` display the FLAPS User's Manual (latest version of this manual)

`apex matview` graphically display a matrix created by FLAPS in a file with a `.mm` extension (see the `print` command, §10.16)

`apex vis` plot data in an ESA-formatted file [34] [15].

1.3.2 How to Avoid Reading this Manual

A quick way to get started is to look at some of the FLAPS demonstration files (chapter 9) and modify them to suit your purposes. At the command line if you type `apex demo` you will change directories to where the demo files are located; type `exit` to return to your previous directory. All files ending with `.ax` are demonstration input files. You can copy them to your own directory to modify and run them by typing, for example ($ is the default Unix command prompt):

```
$ apex stab1 >out 2>err
```

Note that flaps assumes a default file extension of `.ax` so it is not necessary to include it. In this example, output has been redirected to a file named `out` and the standard error listing has been redirected to a file named `err`.

1.4 Getting Help

Like the rest of FLAPS, this manual is a work in progress; it is known to be incomplet, inkorrect, and badly forma^{tt}ed. With that in mind, it is usually a good idea to view the latest version of this manual online by typing at the Unix command prompt:

```
$ apex help
```

or

```
$ apex rtfm
```

which displays a pdf version of the manual. You can then print a copy from within the pdf viewer.

The Loads and Dynamics Wiki (`http://fapa.ca.boeing.com/ladwiki`) has an FLAPS tutorial, FLAPS issues, and lots of information on topics related to Loads and Dynamics. As with all wikis you are encouraged to add content to the web pages.

If you think you have found a bug in FLAPS you should check the bug-tracking system at `http://apex/bugs` to see if your bug has already been reported. If it has not, you can start a new ticket and your problem will get worked on as soon as possible. You will be notified by email through the bug-tracker about progress on your bug.

A mail list is available at `http://apex/flutter` for discussion of a broader range of issues relating to flutter. Mail lists work like this: when you are subscribed to a list you get mail periodically from other members of the list who wish to make comments, ask questions, or report problems about anything related to flutter (not necessarily FLAPS). You can send mail to the list by mailing to `flutter@apex` which then gets sent to all members of the list; in addition your mail gets saved in an archive. You can view all postings to the flutter list by visiting `http://apex/flutter`.

If you want to talk to a real person about FLAPS issues, call, email or visit:

`Calvin J. Lee`

`MS 03-KR`

`425-266-9889`

`40-88 Bldg 2H3-4.4`

`calvin.j.lee@boeing.com`

Chapter 2

The Flaps Input File

FLAPS has three modes of operation: *interpretive mode*, *interactive mode*, and *batch mode*. Interpretive mode is begun when flaps is typed with no filename. In this mode, commands are interpreted and executed as they are typed. Interpretive mode is mainly useful for situations where a small amount of typing is involved; for example to `restore` a savefile and `print` a matrix.

Interactive and batch modes always have an associated *input file* which may be an actual file or may come from the standard input, usually by means of a Unix *here document* [16][29]. A here-document begins with the `<<` operator followed by a string of characters of your choice, and ending with a line containing only that string. For example

```
apex <<eof
   import {myfile}
eof
```

Here-documents are convenient because the input to FLAPS can be placed in the script without creating a separate file, and environment variables are substituted, so a script like

```
export DIR=/home/me
apex <<@
   import { $DIR/myfile }
@
```

is equivalent to

```
apex <<@
   import { /home/me/myfile }
@
```

An FLAPS input file consists of a *control program* followed by any number of optional *data block*s.

2.1 Control Program

The FLAPS control program is always the first set of statements in the input file. It consists of FLAPS commands followed by a line containing only the word `end`. Subsequent lines are ignored up to the beginning of the first data block. FLAPS commands are documented in chapter 10. Commands have the form

command { *options*

...

}

where *whitespace* (spaces, tabs, and newlines) are ignored, *command* is one of the commands listed in chapter 10, and the options are enclosed between curly braces. Commas separate options on a line; the last option on a line does not need to be followed by a comma - the line break serves to separate it from the next option. **There is no limit on the length of lines**.

Comments may be included anywhere by including a number sign (#) followed by the comment. The # does not have to be the first character on the line. Anything up to the # is taken as part of the control program; the rest of the line is ignored. Whitespace is ignored within option lists, and you may include comments; for example, the following is a legal option list:

```
   stab {
     id=pk,             # neutral-stability
#    mode=1             # track mode one or...
     mode=2             # ... track mode two
     ...
```

Note the use of the number sign to disable the `mode=1` option.

Some control constructs from the C or C++ programming languages may be included; for example to test if restoring a savefile was successful you could do this:

```
   if (!restore {savefile}) {
      restore{anothersavefile}
   }
```

where here the exclamation point means *not*, so if restoring `savefile` is unsuccessful an attempt is made to restore `anothersavefile`.

Unix commands may be executed from within the control program by starting the line with an exclamation point (!). The command may be anything you can type on the command line including pipes, redirection of input and output, and background processing (&). For example, to list all directories from within the control program:

```
! ls -l | grep '^d'
```

2.1.1 Environment Variables

Environment variables may be used in the control program similar to the way they are used in shell scripts, by preceding the variable name with a dollar sign:

```
restore { $AXROOT/demo/stab123.sf }
```

In this example `AXROOT` is an environment variable that is defined by FLAPS as the top-level directory of FLAPS. Other environment variables that are defined by FLAPS are

`AXROOT` The top-level directory of an FLAPS installation, for example `/boeing/sw/apex/r6.1`.

`AXTMP` Path of the temporary directory for a running FLAPS job. The lowest-level directory has a name composed of `AxTmp` followed by the process id of the control program to ensure a unique directory name; for example `/boeing/sw/apex/r6.1/demo/stab1.d/AxTmp2144`.

`AXNSIG` The number of significant figures used in comparing floating-point matrix id attributes (§2.3).

Existing environment variables can be changed and new ones created in an FLAPS control program with the `env` option in the `output` command (§10.14). It is not recommended that you change environment variables `AXROOT` or `AXTMP` unless you know what you are doing.

It is also sometimes necessary (but always correct) to enclose the name in curly braces, for example

```
restore { ${AXROOT}/demo/stab123.sf }
```

There is a subtle but important difference between single quotes and double quotes enclosing environment variables, having to do with when the variables are evaluated. If the variable is not quoted, or is enclosed in double quotes it is evaluated by FLAPS when the control program is read. On the other hand, if a variable is enclosed in single quotes it is not evaluated until it is used. For example

```
output { env="AXPATH=${AXPATH}:/proj/bin" }
  ...
output { env='AXPATH=${AXPATH}:/proj/local/bin' }
  ...
output { env="AXPATH=${AXPATH}:/ots/matlab/bin" }
```

After the first `output` statement environment variable `AXPATH` is `/proj/bin` (assuming it is undefined up to this point), after the second it is `/proj/bin:/proj/local/bin`, but after the third it is `/ots/matlab/bin` due to the use of double quotes instead of single quotes.

If the control program is input to FLAPS as a here-document (§2.7) environment variables will be evaluated by the shell as they are fed to FLAPS unless the end marker is enclosed in quotes:

```
apex << '@'
   output { env='AXPATH=${AXPATH}:/proj/local/bin' }
@
```

2.2 Data Blocks

When used in batch mode the input file may contain the control program followed by any number of named data blocks. A data block may contain anything between the opening line which has the form

```
name {{
```

and the ending line which consists of two curly braces:

```
}}
```

That is, a line consisting of a text string which names the block, followed by any number of spaces, followed by two consecutive left curly braces begins a data block, and a line containing just two consecutive curly braces ends the data block

Data blocks can be accessed from anywhere within an FLAPS job as if it were a local file. Using data blocks instead of local files has some important advantages:

- The data can be together in the same file with the rest of the FLAPS job.
- The data block is kept on a private directory for the duration of the job so that it cannot be used inadvertently by another job.
- The private directory where it is kept is removed when the job finishes, so data storage is not an issue.

2.3 Matrix IDs

Throughout this document the term *matrix id* or *mid* for short, is used to refer to the name and attributes associated with a matrix in the FLAPS data manager. In general, an FLAPS matrix consists of a name and a set of optional attribute/value pair. The notation used to specify a matrix name and its attribute/value pair, called a matrix id, has the general form

```
"name,attribute1=value1,attribute2=value2,..."
```

If a mid contains commas it is usually necessary to enclose it in quotes (single or double) in an FLAPS control program to distinguish the commas from those that separate other options. There is no restriction on the lengths of the name, attributes, or values. Floating-point attributes (attributes with floating-point or complex values) do not need to use any particular format for the value; values are compared as floating-point numbers, so for example

```
"genforce,rfi=0.01"
```

is equivalent to

```
"genforce,rfi=1e-2"
```

Floating-point attribute values are compared to 6 significant figures, so for example

```
"genforce,rfi=0.01"
```

is equivalent to

```
"genforce,rfi=0.01000001"
```

The number of significant figures used in comparisons can be modified by setting environment variable `AXNSIG`.

It is not always necessary to include all attributes of a mid. A trailing *ellipsis* can be used to indicate *any attributes*, for example

```
print { "genforce,aerocond=2,..." }
```

will print all matrices with the name `genforce` with the attribute `aerocond` with the value 2, and any other attributes; so this might be equivalent to

```
print {
   "genforce,aerocase=3,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
   "genforce,aerocase=3,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
   "genforce,aerocase=4,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
   "genforce,aerocase=4,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
   "genforce,aerocase=5,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
   "genforce,aerocase=5,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
   "genforce,aerocase=6,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
   "genforce,aerocase=6,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
   "genforce,aerocase=7,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
   "genforce,aerocase=7,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
}
```

In some cases specifying only one or more attributes suffices; for example to save all matrices with the attribute `id=pk` the command

```
save { id=pk }
```

would suffice. Or, in the example above

```
print { aerocond=2 }
```

prints all the `genforce` matrices with the attribute `aerocond=2`.

Some FLAPS modules will automatically add the trailing ellipsis when fetching matrices if fetching the specified name fails. For example

```
alge { gaf = 2*genforce }
```

will multiply all matrices with the name `genforce` and any attributes by 2.

2.3.1 Case Sensitivity

As are most things in FLAPS matrix names are case sensitive: `genforce`, `Genforce`, and `GENFORCE` are all different names.

2.4 What's in a Number?

Floating point numbers in the FLAPS control program have a very flexible format; the following numbers are equivalent:

```
314
314.0
0.0314e+4
31400.00D-2
.314E+3
```

A complex number u + iv may be represented as either (u + iv) or (u + vi). For example, the following are valid complex numbers:

```
(3.14+i6.28)
(0.0314e+2 + i.628d+1)
(314D-2 + 0.628e+1i)
```

2.5 Pre-Defined Constants and Conversions

For convenience several constants and conversion factors are defined. They are used primarily to scale matrices with the `alge` command (§10.5) or as a shorthand for parameter conversion factors (§5.1).

`HZPRS` conversion factor from rad/sec to Hz ($1/2\pi$)

`RSRPM` conversion factor from rad/sec to revolutions/min (rpm). ($\frac{2\pi}{60}\frac{rad/sec}{rpm}$)

`KPIPS` conversion factor from inches/second to knots (nautical-miles/hour)

$$\frac{3600 \text{ sec/hr}}{(12 \text{ in/ft})(6076.11549 \text{ ft/naut mi})} = 0.0493736500 \text{ knots/(in/sec)} \tag{2.1}$$

`DPR` conversion factor from radians to degrees ($180/\pi$)

`RPD` conversion factor from degrees to radians ($\pi/180 = 0.01745...$)

`G` acceleration of gravity = $386.0885826 in/sec^2$. Also, according to Newton's second law `G` can be considered a conversion factor from pounds-force to pounds-mass: $386.0885826 \frac{lb_m in}{lb_f sec^2}$

`PI` $\pi = 3.14159265358979323846264 33....$

2.6 Curly Braces

You have probably noticed by now the extensive use of curly braces ({ and }) to enclose options and blocks of data. This reflects the fact that FLAPS is written in the C++ programming language, which uses curly braces, as do many modern programming languages.

A common error when writing FLAPS input files is to leave out or misplace a curly brace; the easy way to check for matching opening and closing braces is with the matching capability of your editor. If you use the vi editor just put the cursor over either the opening or closing brace, press the percent key and the cursor will move to the matching brace. The same works for parentheses and square brackets. Depending on the version of `vi` and the options you use it may also highlight the matching brace. `emacs` has a similar capability using the ctrl-alt-n and ctrl-alt-p key combinations.

2.7 Running Flaps

FLAPS has three modes of operation: *interpretive mode*, *interactive mode*, and *batch mode*. Interpretive mode is begun when flaps is typed with no filename. In this mode, commands are interpreted and executed as they are typed. Interpretive mode is mainly useful for situations where a small amount of typing is involved; for example to `restore` a savefile and `print` a matrix.

Interactive and batch modes always have an associated *input file* which may be an actual file or may come from the standard input, usually by means of a Unix *here document* [16][29]. A here-document begins with the `<<` operator followed by a string

of characters of your choice, and ending with a line containing only that string. For example

```
apex <<eof
   import {myfile}
eof
```

Here-documents are convenient because the input to FLAPS can be placed in the script without creating a separate file, and environment variables are substituted, so a script like

```
export DIR=/home/me

apex <<@

   import { $DIR/myfile }

@
```

is equivalent to

```
apex <<@

   import { /home/me/myfile }

@
```

If you do not want environment variable to be evaluated when they are input to FLAPS enclose the end marker in quotes:

```
export DIR=/home/me
apex << '@'
   output { env='DIR=$DIR/tmp' }
   import { '$DIR/myfile' }      # imports from /home/me/tmp
@
```

Chapter 3

Equations of Structural Dynamics

This chapter discusses some of the equations of motion which occur in this manual and introduces some terminology used. It is written from the point of view of a practicing engineer rather than from a theoretical point of view. Many textbooks focus on the theory of structural dynamics, for example [5], [10], [21], and [12]. Instead, we begin not with Lagrange's equations but with the equations of motion for a structure modeled with the finite element method, the most widely used modeling technique today. The finite element method (fem) had its origins at the Boeing Company with a paper co-authored by the former head of Flutter Research, M.J. Turner [51]. [1]

3.1 Finite Elements

The finite element method is a technique for representing a continuum (with an infinite number of degrees of freedom) with a finite number of *nodes* with the displacements between the nodes represented by interpolating polynomials. The basic equations of motion for a structure discretized by the finite element method are

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{G}\dot{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{x} = \boldsymbol{f}(t) \tag{3.1}$$

where the mass matrix ($\boldsymbol{M}$), the gyroscopic matrix ($\boldsymbol{G}$), and the stiffness matrix ($\boldsymbol{K}$) are large and sparse (see §3.10 for other properties of these matrices). Rows and columns of these matrices correspond to displacements and rotations at the nodes, generally termed *nodal degrees of freedom* (dof). With a given set of initial conditions and forcing function these equations can be integrated to give the discrete displacements $\boldsymbol{x}(t)$. This is known as a *time-domain* analysis.

If the mass, gyro, and stiffness matrix are not functions of time or the displacements, equation 3.1 is *linear*; otherwise it is *nonlinear*. Linear equations can be treated with simplification techniques such as characteristic equations (§3.3). Nonlinear equations

[1] It is ironic that the Boeing Company was never able to capitalize on this seminal work

are much more difficult, but can sometimes be simplified, using for example *describing functions* (appendix G).

3.1.1 Reducing Finite Element Model Size

The matrices in equation 3.1 are often very large and contain much more detail than necessary for most stability and response analyses. Reducing the size of these matrices is important because computational costs for stability and response analyses rise with the cube of matrix size; reducing the matrix size by half reduces the cost of stability analysis by a factor of eight. Reduction techniques include static reduction, Guyan reduction, load basis, and truncated generalized coordinates.

Static Reduction

The number of degrees of freedom in a finite element model (substructured or not) is often reduced using a technique known as *static reduction* or *static condensation*. Degrees of freedom which are to be retained are specified with an `a-set` in NASTRAN or `RETAINED` freedoms in ATLAS `BC DATA`; those which are to be eliminated are in the NASTRAN `o-set` or the ATLAS `FREE` freedoms in ATLAS `BC DATA`. Partitioning a stiffness matrix into retained and eliminated degrees-of-freedom the stiffness matrix can be written as

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{rr} & \boldsymbol{K}_{re} \\ \boldsymbol{K}_{er} & \boldsymbol{K}_{ee} \end{bmatrix} \tag{3.2}$$

Then assuming there are no external forces on the nodes to be eliminated:

$$\boldsymbol{K}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{K}_{rr} & \boldsymbol{K}_{re} \\ \boldsymbol{K}_{er} & \boldsymbol{K}_{ee} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x}_r \\ \boldsymbol{x}_e \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{f}_r \\ \boldsymbol{0}_e \end{Bmatrix} \tag{3.3}$$

and from this we can solve for the retained dof:

$$\left(\boldsymbol{K}_{rr} - \boldsymbol{K}_{re}\boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er}\right)\boldsymbol{x}_r = \boldsymbol{f}_r \tag{3.4}$$

which is equivalent to a change of coordinate basis:

$$\boldsymbol{x} = \boldsymbol{\Phi}\boldsymbol{q} \tag{3.5}$$

where

$$\begin{aligned} \boldsymbol{q} &= \boldsymbol{x}_r \\ \boldsymbol{\Phi} &= \begin{bmatrix} \boldsymbol{I} \\ -\boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er} \end{bmatrix} \end{aligned} \tag{3.6}$$

Guyan Reduction

Guyan reduction [18] applies the same transformation used in static condensation to the mass matrix in addition to the stiffness matrix:

$$\begin{aligned}
\hat{\boldsymbol{M}} &= \boldsymbol{\Phi}^{\boldsymbol{t}}\boldsymbol{M}\boldsymbol{\Phi} \\
&= \boldsymbol{M}_{rr} - \left[\boldsymbol{M}_{re}\boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er} + \left(\boldsymbol{M}_{re}\boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er}\right)^{t}\right] + \left(\boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er}\right)^{t}\boldsymbol{M}_{ee}\boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er} \\
\hat{\boldsymbol{K}} &= \boldsymbol{\Phi}^{\boldsymbol{t}}\boldsymbol{K}\boldsymbol{\Phi} \\
&= \boldsymbol{K}_{rr} - \boldsymbol{K}_{er}^{t}\boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er} \\
\hat{\boldsymbol{f}} &= \boldsymbol{\Phi}^{t}\boldsymbol{f} \\
&= \boldsymbol{f}_{r} - \boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er}\boldsymbol{f}_{e}
\end{aligned} \tag{3.7}$$

Recall the assumption made with static condensation was that the nodes to be eliminated had no forces on them; extending this transformation to the mass matrix implies that the eliminated dof have no inertia forces on them. In other words Guyan reduction can be viewed as a way of redistributing the masses from the eliminated dof to the retained dof.

A related technique used in Elfini is called a *loads basis*, obtained by applying unit forces on the dof to be retained and computing the resulting displacements.

$$\boldsymbol{K}\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{K}_{rr} & \boldsymbol{K}_{re} \\ \boldsymbol{K}_{er} & \boldsymbol{K}_{ee} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Phi}_{rr} \\ \boldsymbol{\Phi}_{er} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{rr} \\ \boldsymbol{0}_{er} \end{bmatrix} \tag{3.8}$$

solving for $\boldsymbol{\Phi_{er}}$ is easy:

$$\boldsymbol{\Phi}_{er} = -\boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er}\boldsymbol{\Phi}_{rr} \tag{3.9}$$

Instead of solving for $\boldsymbol{\Phi_{rr}}$ note that because we do not want to transform the retained dof, $\boldsymbol{\Phi_{rr}}$ must be the identity, so normalize the basis vectors by post-multiplying by $\boldsymbol{\Phi_{rr}^{-1}}$:

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_{rr} \\ \boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er}\boldsymbol{\Phi}_{rr} \end{bmatrix} \boldsymbol{\Phi}_{rr}^{-1} = \begin{bmatrix} \boldsymbol{I}_{rr} \\ \boldsymbol{K}_{ee}^{-1}\boldsymbol{K}_{er} \end{bmatrix} \tag{3.10}$$

which is identical to equation 3.6.

Guyan Reduction Reconsidered

These techniques are usually applied to finite element models prior to a free-vibration eigensolution and a further reduction to a truncated set of generalized coordinates 3.2. In the past this was an important step to make the eigensolution affordable; however modern eigenvalue solution techniques account for the sparsity of mass and stiffness matrices and Guyan reduction destroys sparsity so an eigensolution can actually be more expensive after Guyan reduction. This, together with the fact that Guyan reduction is an approximation technique makes it less useful than in the past.

3.2 Generalized Coordinates

There are two aspects to describing the behavior of dynamic (or static) systems: a set of coordinates and their associated *coordinate basis*. Structures modeled with the finite element method usually use Cartesian coordinate systems with 3 mutually perpendicular axes as the coordinate basis with translations along and rotations about the axes as the coordinates. Other, less common coordinate systems used in the finite element method are spherical and cylindrical coordinates, also referred to as nodal coordinates and nodal degrees of freedom because each dof represents specific motion of a single node in the FEM.

Nodal coordinates are not the only way to represent the behavior of structures; generalized coordinates, as the name implies are any set of coordinates and their associated bases that are capable of any displacements the original nodal coordinates are capable of. For example, one of the most widely used sets of generalized coordinates are based on free-vibration eigenvectors. A model with `n` dof has `n` free-vibration eigenvectors; if all these are used as the generalized coordinate basis the model in these generalized coordinates will also have `n` dof and is capable of reproducing any set of displacements in the nodal dof.

The matrices in equation 3.1 are often very large and contain much more detail than necessary for most stability and response analyses. Reducing the size of the matrices is usually done by introducing a *truncated* generalized coordinate basis consisting of a set of vectors which in some sense approximate solutions to the equations of motion, for example a few low-frequency free-vibration eigenvectors. These vectors are often referred to as *modes*, not to be confused with *aeroelastic modes* (§7.1). The degrees of freedom corresponding to the generalized-coordinate basis vectors are known as *generalized coordinates*. Transforming the equations of motion to the new basis vectors is simply a matter of pre- and post-multiplying each matrix by the basis vectors:

$$\begin{aligned} \hat{\boldsymbol{M}} &= \boldsymbol{\Phi}^{\boldsymbol{t}} \boldsymbol{M} \boldsymbol{\Phi} \\ \hat{\boldsymbol{G}} &= \boldsymbol{\Phi}^{\boldsymbol{t}} \boldsymbol{G} \boldsymbol{\Phi} \\ \hat{\boldsymbol{K}} &= \boldsymbol{\Phi}^{\boldsymbol{t}} \boldsymbol{K} \boldsymbol{\Phi} \\ \hat{\boldsymbol{f}} &= \boldsymbol{\Phi}^{t} \boldsymbol{f} \end{aligned} \tag{3.11}$$

where $\boldsymbol{\Phi}$ is the matrix of generalized-coordinate basis vectors, which typically has many more rows than columns and each column is a basis vector. These matrices are known as the *generalized mass* and *generalized stiffness* respectively, and the *generalized force* vector.

The generalized coordinates are related to the finite-element displacements through the basis vectors:

$$\boldsymbol{x} = \boldsymbol{\Phi} \hat{\boldsymbol{x}} \tag{3.12}$$

where $\hat{\boldsymbol{x}}$ is the vector of generalized-coordinates, sometimes referred to as *participation factors* because they are the amount that each of the basis vectors participate in sub-

sequent solutions of the equations of motion. With this transformation, equation 3.1 becomes

$$\hat{\boldsymbol{M}}\ddot{\hat{\boldsymbol{x}}} + \hat{\boldsymbol{G}}\dot{\hat{\boldsymbol{x}}} + \hat{\boldsymbol{K}}\hat{\boldsymbol{x}} = \hat{\boldsymbol{f}}(t) \tag{3.13}$$

The choice of basis vectors is very important to successfully reducing the size of the problem; a poor choice will result in a loss of accuracy. The key is to ensure that some combination of the basis vectors will result in a reasonably accurate approximation to the true solution vector. Flutter solutions are generally approximated very well with *free-vibration modes* as basis vectors. This is due to the fact that air loads tend to be distributed over the entire structure without highly localized forces; conversely dynamic response problems that have large point loads tend to have solutions which do not resemble free-vibration modes, so a free-vibration basis usually must be enriched with solutions involving the point load. Various techniques based on this idea go by the names *mode acceleration* and *force summation* [10][5].

3.2.1 Creating Basis Vectors

There are numerous techniques for creating basis vectors; among the most common in use are free-vibration modes, *assumed modes*, *superelement modes*, and *branch modes*.

Free-vibration modes are computed by solving the generalized eigenvalue problem (§**??**)

$$\boldsymbol{K}\boldsymbol{\phi}_i = \omega_i^2 \boldsymbol{M}\boldsymbol{\phi}_i \tag{3.14}$$

for a small number of the lowest-frequency (ω) modes and using them to form a basis $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1 \boldsymbol{\phi}_2 ...]$

Assumed modes are sets of displacements that are created based on some special knowledge of the behavior of the structure. They generally involve only a few nodal degrees-of-freedom. For example a common use for assumed modes is to include rigid rotations of control surfaces in a model that does not have that mode in a set of free-vibration modes due to the presence of a stiff actuator.

Superelement modes (also known as component or modal-synthesis modes) (§H.2.1) result from splitting a structure into superelements and computing free-vibration modes of each piece separately. The assembled set of modes is generally larger to achieve the same accuracy as free-vibration modes, so less reduction in the problem size is possible. However, each superelement can be modified and analyzed independently.

The Branch Modes technique (§H.2.2) is a modification of the technique used to compute superelement modes which isolates superelements in the sense that a superelement has no stiffness coupling between it and the rest of the structure. This allows the stiffness of the superelement to be varied independent of the rest of the structure for stiffness parameter studies. The modes produced by the Branch Modes method have displacements only in the superelement; the rest of the structure is fixed.

3.3 Frequency Domain: Characteristic Equations

Up to this point the equations of motion have been expressed as functions of time; here we show how the equations may be transformed to the *frequency domain*, replacing time-dependence with frequency-dependence. In terms of solution efficiency, this is perhaps the most important simplification that can be made to the equations of motion.

If the right-hand side of eqn. 3.13 is a harmonic function, that is if it has the form $\boldsymbol{f}(t) = \mathrm{Re}\left(\hat{\boldsymbol{f}}e^{st}\right)$ where $s = \sigma + i\omega$ is the *characteristic exponent* or *Laplace variable*, $\hat{\boldsymbol{f}}$ is a constant complex vector and Re is the real part of a complex number, we can assume a solution of the form

$$\begin{aligned}\boldsymbol{x}(t) &= \mathrm{Re}\left(\boldsymbol{q}e^{st}\right)\\ &= \mathrm{Re}\left[\boldsymbol{q}e^{\sigma t}\left(\cos\omega t + i\sin\omega t\right)\right]\\ &= e^{\sigma t}\left(\boldsymbol{q}^r\cos\omega t - \boldsymbol{q}^i\sin\omega t\right)\end{aligned} \tag{3.15}$$

where $\boldsymbol{q} = \boldsymbol{q}^r + i\boldsymbol{q}^i$ is a complex vector of generalized coordinate amplitudes. The k^{th} component can be written in various ways[9]:

$$q_k = \mathrm{Re}\left(q_k\right) + i\,\mathrm{Im}\left(q_k\right) = |q_k|e^{i\phi_k} = |q_k|\left(\cos\phi_k + i\sin\phi_k\right) \tag{3.16}$$

where $\phi_k = \arctan\frac{\mathrm{Im}(q_k)}{\mathrm{Re}(q_k)}$ is the *phase angle* associated with the k^{th} element of the complex vector $\boldsymbol{q}$ and $|q_k| = \sqrt{\left(q_k^r\right)^2 + \left(q_k^i\right)^2}$ is the *absolute value*. Then

$$\begin{aligned}x_k(t) &= \mathrm{Re}|q_k|e^{i\phi_k}e^{st} = \mathrm{Re}|q_k|e^{\sigma t}e^{i(\omega t+\phi_k)}\\ &= \mathrm{Re}\left[|q_k|e^{\sigma t}\left[\cos(\omega t+\phi_k) + i\sin(\omega t+\phi_k)\right]\right]\\ &= |q_k|e^{\sigma t}\cos(\omega t+\phi_k)\end{aligned} \tag{3.17}$$

Substitution into 3.13

$$\left[s^2\hat{\boldsymbol{M}} + s\hat{\boldsymbol{G}} + \hat{\boldsymbol{K}}\right]\boldsymbol{q}e^{st} = \hat{\boldsymbol{f}}e^{st} \tag{3.18}$$

and dividing both sides by e^{st} results in *characteristic equations* of motion: [2]

$$\left[s^2\boldsymbol{M} + s\boldsymbol{G} + \boldsymbol{K}\right]\boldsymbol{q} = \boldsymbol{f} \tag{3.19}$$

With no forcing function on the right-hand side this reduces to a real generalized eigenvalue problem also known as the *free-vibration eigenvalue* problem

$$\left[-\omega^2\boldsymbol{M} + \omega\boldsymbol{G} + \boldsymbol{K}\right]\boldsymbol{\phi} = \boldsymbol{0} \tag{3.20}$$

[2] without loss of generality we drop the hats from this point on: subsequent equations can be in terms of generalized-coordinates or the original physical degrees-of-freedom.

Equation 3.19 are said to be in the *frequency* or *Laplace* domain because the time variable has been replaced with the frequency dependent variable s.

The motion of the structure is determined by the characteristic exponent. From eqn. 3.17 it can be seen that the frequency of oscillation is determined by ω; furthermore, σ determines whether oscillations are growing (positive values), decaying (negative values) or are *neutrally stable*, neither growing nor decaying (zero values).

Related to the real part of the characteristic exponent σ is the *log decrement*, defined as the log of the ratio of two consecutive amplitudes of a harmonic motion:

$$\delta = ln\left[\frac{x(t)}{x(t+\frac{2\pi}{\omega})}\right] = ln\left[\frac{e^{\sigma t}}{e^{\sigma(t+\frac{2\pi}{\omega})}}\right] = -\frac{2\pi\sigma}{\omega} = -\pi\gamma \tag{3.21}$$

where

$$\gamma = \frac{2\sigma}{\omega} \tag{3.22}$$

is the *growth rate*. Calling this quantity growth rate is more semantically correct than the traditional term *decay rate* because positive values cause growing amplitudes of oscillation, not decaying. As shown by equation 3.29, values of growth rate are close to values of structural damping with the opposite sign, and this gives a way of measuring structural damping in a structure: excite the structure in a particular free-vibration mode, measure consecutive amplitudes and compute the log decrement.

3.4 Damping

All materials exhibit a loss of energy as they are deformed according to the laws of thermodynamics. In structural-dynamic analyses these losses are typically modeled as structural damping or viscous damping. Of these structural damping is more often used explicitly and can be computed from vibration tests (§3.3); viscous damping is implicit in unsteady aerodynamics, but is harder to measure in structures.

3.4.1 Structural Damping

Structural damping consists of internal friction due to hysteresis, and friction between components of a structure [32][50]. Internal friction is independent of the rate of deformation, or of the frequency of oscillation. Moreover, it is proportional to the amplitude of displacement and acts opposite to the velocity. The effects of friction between components complicate the measurement of structural damping, and usually result in structural damping which is not linear with the amplitude of displacement. Still it is useful to represent structural damping by $id\boldsymbol{K}$ where $i^2 = -1$ and d is the structural damping coefficient:

$$\left[s^2\boldsymbol{M} + (1+id)\boldsymbol{K}\right]\boldsymbol{q} = \boldsymbol{f} \tag{3.23}$$

Structural damping coefficients are estimated during ground vibration tests. They are different for each mode of vibration but generally range in value from 0.005 to 0.03,

depending upon the amplitude of vibration. Because of the uncertainty associated with structural damping, analyses are often done at zero and 0.03 to assess the effects of this uncertainty on the solution. This is reflected in the FAA requirements for flutter clearance.

When the generalized coordinates in an analysis are based on vibration modes it is often desirable to use different structural damping coefficients for each mode; this results in a stiffness matrix that is a function of multiple structural damping coefficients:

$$\boldsymbol{K} = \boldsymbol{K}(d_k), \qquad k = 1, m \tag{3.24}$$

Structural damping coefficients for individual degrees of freedom can be specified with the FLAPS param command (§10.15); be aware of the interaction between individual coefficients and the overall structural damping coefficient when using both types (§6.3.2).

Small values of structural damping result in growth rates with values nearly the negative of the structural damping coefficient. To see this, consider a single degree of freedom vibration problem with structural damping:

$$\left[s^2 M + (1 + id)K\right] x = 0 \tag{3.25}$$

$$\sigma^2 + 2i\sigma\omega - \omega^2 = -\frac{(1 + id)K}{M} \tag{3.26}$$

Equating the real and imaginary parts

$$\sigma^2 - \omega^2 = -\frac{K}{M} \tag{3.27}$$

$$d = -\frac{2\sigma\omega M}{K} = -\frac{2\sigma\omega}{\omega^2 - \sigma^2} \tag{3.28}$$

which for motion that is nearly pure harmonic (oscillations that are neither growing nor decaying rapidly) ($\sigma^2 \ll \omega^2$)

$$d \approx -\frac{2\sigma}{\omega} = -\gamma \tag{3.29}$$

This relationship, derived for a single uncoupled equation with small damping will break down with larger values of damping and possibly with equation coupling; nevertheless this equivalence can be used to estimate the effects of structural damping on a structure without damping: the point where growth rate is, for example 0.03 is close to the point where the structure would be neutrally stable if it had 0.03 structural damping.

There are two ways to include structural damping in an analysis: with the `sdamp` parameter (§5.5), which applies to the entire stiffness matrix, or with user-defined parameters which apply to one or more degrees of freedom in the stiffness matrix, defined with the param command.

3.4.2 Viscous Damping

As the name implies, viscous damping is associated with fluid viscosity and is proportional to the velocity, or frequency for harmonic oscillations, thus the viscous damping matrix $\boldsymbol{V}$ is multiplied by the characteristic exponent:

$$\left[s^2\boldsymbol{M} + s\boldsymbol{V} + s\boldsymbol{G} + (1+id)\boldsymbol{K}\right]\boldsymbol{q} = \boldsymbol{f} \tag{3.30}$$

Viscous damping may be included in an analysis with the `vdamp` option.

3.5 Unsteady Aerodynamics

Unsteady aerodynamic matrices are usually generated in terms of the generalized-coordinate basis $\boldsymbol{\Phi}$ as functions of Mach number and reduced frequency. The Mach number is defined as

$$M = \frac{V_t}{a} \tag{3.31}$$

where V_t is the true airspeed (the speed relative to the earth) and a is the speed of sound in the fluid (usually the standard atmosphere). Reduced frequency is defined as

$$p = \frac{sb}{V_t} = \frac{\sigma b}{V_t} + i\frac{\omega b}{V_t} = g + ik \tag{3.32}$$

where k is often referred to as *k-value* or simply reduced frequency, and p is usually referred to as the *complex k-value* or *complex reduced frequency*. b is a reference length, necessary to make the reduced-frequency dimensionless. Here we assume the reference length is one and do not consider it further.

By convention the unsteady aerodynamic matrix is included in the characteristic equations of motion scaled by the negative of the dynamic pressure:

$$\left[s^2\boldsymbol{M} + s(\boldsymbol{V}+\boldsymbol{G}) + (1+id)\boldsymbol{K} - q\boldsymbol{Q}(p,M)\right]\boldsymbol{q} = \boldsymbol{f} \tag{3.33}$$

Some unsteady aerodynamic processors are capable of generating matrices that are functions of complex reduced frequency and Mach number; others (including NASTRAN [41]) can generate matrices only at imaginary values of reduced frequency (k) and Mach number. It has been shown [13, 14] that linear unsteady aerodynamic theories may be extended to complex reduced frequencies simply by replacing the pure imaginary reduced frequency with complex reduced frequency. This is the approach taken in the doublet-lattice processors in ATLAS and FLAPS. Using aerodynamics generated at values of complex reduced frequency would then give correct results for motions that are growing or decaying.

Alternatively, over the years there have been several attempts to approximate unsteady aerodynamic forces for decaying motion, in addition to the usual forces generated for harmonic motion. Here we give details on two approximations that are available in the FLAPS `stab` command.

3.5.1 NASTRAN

NASTRAN [41] adds a term dependent on growth rate:

$$\begin{aligned} \boldsymbol{Q}(p) &\approx \boldsymbol{Q}(k) + \frac{\sigma}{\omega}\mathrm{Im}(\boldsymbol{Q}(k)) \\ &= \boldsymbol{Q}(k) + \frac{\gamma}{2}\mathrm{Im}(\boldsymbol{Q}(k)) \end{aligned} \tag{3.34}$$

where $\mathrm{Im}(\boldsymbol{Q})$ is the imaginary part of the (real) reduced frequency interpolated unsteady aerodynamic matrix.

When the frequency approaches zero this added term can become large, depending on the value of σ. so it would seem to give questionable results at low frequencies, but this does not appear to be the case (see §3.5.3).

3.5.2 g-Method

Another technique for modifying unsteady aerodynamic matrices generated at (imaginary) reduced frequency is due to Chen [8]. Using the fact that complex unsteady aerodynamic matrices are analytic functions and therefore satisfy the Cauchy-Riemann equations [9], for small values of g the aerodynamic matrix can be written as

$$\begin{aligned} \boldsymbol{Q}(p) &\approx \boldsymbol{Q}(k) + g\frac{\partial \boldsymbol{Q}}{\partial g} \\ &= \boldsymbol{Q}(k) - ig\boldsymbol{Q}'(k)) \end{aligned} \tag{3.35}$$

where the complex reduced frequency $p = g + ik$, and

$$\frac{\partial \boldsymbol{Q}}{\partial g} = -i\frac{\partial \boldsymbol{Q}}{\partial k} \tag{3.36}$$

due to the Cauchy-Riemann equations. It can be shown [8] that this approximation is the same as the NASTRAN approximation if the aerodynamic matrix is a linear function of reduced frequency or at zero reduced frequency.

3.5.3 Comparison

In both the NASTRAN and g-method modifications of the unsteady aerodynamic matrix the modifications disappear when σ is zero, so these modifications will have no effect on neutral stability. This example shows little difference in result between these three aerodynamic formulations.

Sample problem stab10.ax is from the NASTRAN Test Problem Library. It compares solutions obtained using the p-k method with the NASTRAN and g-method aerodynamic modifications, and with the V-g formulation (§7.1.1). Two modes in this problem diverge; one at 976 knots true airspeed and the other at 2254 knots. Of these only the V-g formulation predicts the correct divergence velocity.

3.6 Controls Equations

Equations representing control-laws may be included in a matrix $\boldsymbol{T}$ which is usually larger than the other matrices in the characteristic equation to account for the controls equations. The general form of a $\boldsymbol{T}$ matrix is

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T}_{11} & \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} & \boldsymbol{T}_{22} \end{bmatrix} \tag{3.37}$$

where the $\boldsymbol{T_{11}}$ block is usually zero, the $\boldsymbol{T_{21}}$ block provides input to the controls equations due to motion of the structure, the $\boldsymbol{T_{12}}$ block provides feedback to the structure, and the $\boldsymbol{T_{22}}$ block comprises terms of the controls equations.

Control-law matrices are usually functions of the characteristic exponent and dynamic pressure; including a controls matrix, the characteristic equations of motion can be written

$$\left[s^2\boldsymbol{M} + s(\boldsymbol{V}+\boldsymbol{G}) + (1+id)\boldsymbol{K} - q\boldsymbol{Q}(p,M) + \boldsymbol{T}(s,q)\right]\boldsymbol{q} = \boldsymbol{f} \tag{3.38}$$

where the mass, stiffness, and aero matrices have been expanded to match the size of the $\boldsymbol{T}$ matrix:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_{11} & 0 \\ 0 & 0 \end{bmatrix} \tag{3.39}$$

as has the vector of generalized coordinates:

$$\boldsymbol{q} = \begin{Bmatrix} \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \end{Bmatrix} \tag{3.40}$$

Controls equations can be described with state-space matrices created using Matlab and Simulink (known as the ABCD approach), or by writing a Fortran subroutine which fills in the $\boldsymbol{T}$ matrix (known as the *user-subroutine* or *coded* approach). More general control-law equations may be represented with user-written subroutines at the cost of more effort on the part of the subroutine writer.

Section 6.2 and appendix C detail the ABCD approach, while section 6.4 discusses the use of user-written subroutines; actual usage instructions are in section 10.15.

3.7 Dynamic Matrix

The sum of all the matrices is known as the *dynamic matrix* $\boldsymbol{D}$:

$$\boldsymbol{D}\boldsymbol{q} = \left[s^2\boldsymbol{M} + s(\boldsymbol{V}+\boldsymbol{G}) + (1+id)\boldsymbol{K} - q\boldsymbol{Q}(p,M) + \boldsymbol{T}(s,q)\right]\boldsymbol{q} = \boldsymbol{f} \tag{3.41}$$

Equation 3.41 is used to solve the two major problems of structural dynamics: stability and response. The homogeneous form (no right-hand-side) is used to determine stability by solving the nonlinear eigenvalue problem (also known as the *flutter equation*)

$$\boldsymbol{D}(s)\boldsymbol{q} = \boldsymbol{0} \tag{3.42}$$

for eigenvalues and eigenvectors $(s, \boldsymbol{q})$. Stability is then determined from the real part of $s = \sigma + i\omega$. If there are n equations in 3.42 there are theoretically n pair of eigenvalues and eigenvectors, collectively referred to as *aeroelastic modes*.

Response analyses solve the system of complex linear equations

$$\boldsymbol{D}(s)\boldsymbol{q} = \boldsymbol{f} \tag{3.43}$$

with $\sigma = 0$, giving the steady-state response to harmonic excitation.

3.7.1 Time Domain Revisited

The dynamic matrix equation (3.41) can be written for time-domain problems by neglecting unsteady aerodynamics and structural damping

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + (\boldsymbol{V} + \boldsymbol{G})\,\dot{\boldsymbol{q}} + \boldsymbol{K} + \boldsymbol{T}(t, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = \boldsymbol{f}(t) \tag{3.44}$$

Unsteady aerodynamics formulated in the frequency domain can be transformed to the time domain (see sect. **??**).

3.8 Further Reduction of Matrix Size

Free-vibration modes may be the simplest modes to compute but they are not the most useful for doing parameter studies. Often it is necessary to include so-called *assumed modes*, which as the name implies are shapes specified by the user, usually to include control-surface rotations as separate degrees of freedom. In addition, generalized coordinates based on superelements (see appendix H). Both of these techniques result in generalized-coordinate bases (modes matrices) which are *uncoupled* in the sense that some degrees of freedom have motion in only one mode (column of $\boldsymbol{\Phi}$). The term *uncoupled* can cause confusion because it is used with opposite meanings depending on whether you are talking about modes or equations. *coupled modes* refers to modes which have been transformed so that they represent free-vibration modes of the entire airplane; any assumed modes or superelement modes have *coupled* with the rest of the structure. Coupled modes result in diagonal generalized mass and stiffness matrices.

On the other hand *coupled equations* means that the mass and/or stiffness matrices are not diagonal - exactly the opposite meaning from coupled modes. Thus the vibration problem 3.14 uncouples the equations of motion by coupling the modes.

The presence of assumed modes and superelement modes increases the size of the matrices over the equivalent free-vibration modes, and sometimes it is desirable to reduce the size by coupling the modes (or uncoupling the equations of motion). This is done by solving the vibration problem 3.14 with the generalized mass and stiffness matrices for a reduced set of modes $\hat{\boldsymbol{\Phi}} = \left[\hat{\phi}_1 \hat{\phi}_2 ...\right]$. Then just as equation 3.12 transformed from generalized coordinates to physical degrees of freedom, the transformation from the reduced generalized coordinates to the original generalized coordinates, and to physical

degrees of freedom is

$$\boldsymbol{x} = \boldsymbol{\Phi q} = \boldsymbol{\Phi}\hat{\boldsymbol{\Phi}}\hat{\boldsymbol{q}} \tag{3.45}$$

The addition of controls equations complicates the transformation somewhat. Assuming we do not want to alter the controls equations the transformation from generalized coordinates to reduced generalized coordinates is

$$\hat{\hat{\boldsymbol{\Phi}}} = \begin{bmatrix} \hat{\boldsymbol{\Phi}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{3.46}$$

which transforms the sensor and feedback blocks of the $\boldsymbol{T}$ matrix (§3.6)

$$\hat{\boldsymbol{T}} = \hat{\hat{\boldsymbol{\Phi}}}^{t} \boldsymbol{T} \hat{\hat{\boldsymbol{\Phi}}} = \begin{bmatrix} \hat{\boldsymbol{\Phi}}^{t} \boldsymbol{T}_{11} \hat{\boldsymbol{\Phi}} & \hat{\boldsymbol{\Phi}}^{t} \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} \hat{\boldsymbol{\Phi}} & \boldsymbol{T}_{22} \end{bmatrix} \tag{3.47}$$

Demo problem stab7.ax shows how you can make this transformation using FLAPS commands.

3.9 Units

The units of all parameters, vectors, and matrices in equation 3.41 must be consistent: each term must result in the same units. Matrix units are very important to be aware of, especially if the matrices used in an analysis come from different sources, for example if the mass and stiffness matrices are from NASTRAN and the unsteady aerodynamic matrices are from TRANAIR.

The default units for the right and left sides of an equation in nodal degrees-of-freedom such as 3.41 are pounds force (lb_f) for translational degrees-of-freedom and lb_f-in for rotational degrees-of-freedom. It is the user's responsibility to ensure that units are consistent if any changes are made to the parameter or matrix units.

Tables 3.1, 3.2, 3.3, and 3.4 show the matrix, vector, and parameter terms with default units. Parameter units shown are those used internally to give consistent equations; parameters may also have *external* units as shown in table 5.1. Note that mass terms have been converted from pounds mass (lb_m) to lb_f-sec^2/in using the conversion factor

$$386.0886 \frac{lb_m\text{-}in}{lb_f\text{-}sec^2} \tag{3.48}$$

which arises from the fact that a lb_f is defined as the gravitational force exerted on a 1 lb_m object on the surface of the earth. Note also that the default measure of angles, *radians*, is not technically a unit due to the fact that it is defined as the angle formed at the center of a circle by two radii enclosing an arc with an arc-length equal to the radius, which is dimensionless. Any angle can be expressed in radians by forming a circle with center at the vertex of the angle and any radius; the angle in radians is the ratio of the arc-length divided by the radius.

Term	Default Units
$\boldsymbol{D}_{tt}, \boldsymbol{K}_{tt}$ or $\boldsymbol{T}_{tt}$	lb_f/in
$\boldsymbol{D}_{rt}, \boldsymbol{K}_{rt}$ or $\boldsymbol{T}_{rt}$	lb_f
$\boldsymbol{D}_{tr}, \boldsymbol{K}_{tr}$ or $\boldsymbol{T}_{tr}$	lb_f/rad
$\boldsymbol{D}_{rr}, \boldsymbol{K}_{rr}$ or $\boldsymbol{T}_{rr}$	lb_f-in/rad
$\boldsymbol{M}_{tt}$	lb_f-sec^2/rad^2-in
$\boldsymbol{M}_{rt}$	lb_f-sec^2/rad^2
$\boldsymbol{M}_{tr}$	lb_f-sec^2/rad^3
$\boldsymbol{M}_{rr}$	lb_f-in-sec^2/rad^3
$\boldsymbol{G}_{tt}$ or $\boldsymbol{V}_{tt}$	lb_f-sec/rad-in
$\boldsymbol{G}_{rt}$ or $\boldsymbol{V}_{rt}$	lb_f-sec/rad
$\boldsymbol{G}_{tr}$ or $\boldsymbol{V}_{tr}$	lb_f-sec/rad^2
$\boldsymbol{G}_{rr}$ or $\boldsymbol{V}_{rr}$	lb_f-in-sec/rad^2
$\boldsymbol{Q}_{tt}$	in
$\boldsymbol{Q}_{rt}$	in^2
$\boldsymbol{Q}_{tr}$	in^2/rad
$\boldsymbol{Q}_{rr}$	in^3/rad
$\boldsymbol{\Phi}_t$	in
$\boldsymbol{\Phi}_r$	$radians$
t subscript refers to translational degrees-of-freedom r subscript refers to rotational degrees-of-freedom	

Table 3.1: Units for Matrices in Nodal Degrees-of-Freedom

3.10 Matrix Properties

Here we discuss various properties of matrices in general and some properties of structural-dynamic matrices in particular.

Matrices are often too large to view as printed numbers, so FLAPS uses a graphical visualization tool called MatView [22] which can be used to look at the overall structure of a matrix or examine individual values. The FLAPS print command (§10.16). uses MatView by default.

Mass matrices and stiffness matrices are *symmetric*: $\boldsymbol{M}_{ij} = \boldsymbol{M}_{ji}$ and because they are symmetric they have real eigenvalues. In addition, mass matrices are *positive definite*: all of their eigenvalues are positive. Stiffness matrices are positive definite if the structure has no rigid-body modes; otherwise they have zero eigenvalues and are positive semi-definite. Mass and stiffness matrices in nodal degrees-of-freedom are *sparse* (many elements are zero) so special sparse-matrix techniques are used to treat the eigenvalue problem 3.14.

Gyroscopic matrices are *skew-symmetric*: $\boldsymbol{G}_{ij} = -\boldsymbol{G}_{ji}$. Skew-symmetric matrices have pure imaginary eigenvalues.

Complex matrices can similarly be classified as *Hermitian*:

$$\overline{\boldsymbol{A}}^t = \boldsymbol{A}^* = \boldsymbol{A} \tag{3.49}$$

that is, the conjugate transposed matrix equals the original matrix. The * superscript

Term	Default Units
$\bar{\boldsymbol{D}}_{tt}, \bar{\boldsymbol{K}}_{tt}$ or $\bar{\boldsymbol{T}}_{tt}$	lb_f-in
$\bar{\boldsymbol{D}}_{rt}, \bar{\boldsymbol{K}}_{rt}$ or $\bar{\boldsymbol{T}}_{rt}$	lb_f-$in - rad$
$\bar{\boldsymbol{D}}_{tr}, \bar{\boldsymbol{K}}_{tr}$ or $\bar{\boldsymbol{T}}_{tr}$	lb_f-in
$\bar{\boldsymbol{D}}_{rr}, \bar{\boldsymbol{K}}_{rr}$ or $\bar{\boldsymbol{T}}_{rr}$	lb_f-in-rad
$\bar{\boldsymbol{M}}_{tt}$	lb_f-sec^2-in/rad^2
$\bar{\boldsymbol{M}}_{rt}$	lb_f-sec^2-in/rad
$\bar{\boldsymbol{M}}_{tr}$	lb_f-sec^2-in/rad^2
$\bar{\boldsymbol{M}}_{rr}$	lb_f-in-sec^2/rad
$\bar{\boldsymbol{G}}_{tt}$ or $\bar{\boldsymbol{V}}_{tt}$	lb_f-in-sec/rad
$\bar{\boldsymbol{G}}_{rt}$ or $\bar{\boldsymbol{V}}_{rt}$	lb_f-in-sec
$\bar{\boldsymbol{G}}_{tr}$ or $\bar{\boldsymbol{V}}_{tr}$	lb_f-in-sec/rad
$\bar{\boldsymbol{G}}_{rr}$ or $\bar{\boldsymbol{V}}_{rr}$	lb_f-in-sec
$\bar{\boldsymbol{Q}}_{tt}$	in^3
$\bar{\boldsymbol{Q}}_{rt}$	in^3-rad
$\bar{\boldsymbol{Q}}_{tr}$	in^3
$\bar{\boldsymbol{Q}}_{rr}$	in^3-rad
t subscript refers to translational degrees-of-freedom r subscript refers to rotational degrees-of-freedom	

Table 3.2: Units for Matrices in Generalized Degrees-of-Freedom

denotes the conjugate transpose, sometimes called the Hermitian transpose. A *skew-Hermitian* matrix is one where the conjugate transpose equals the negative of the matrix: $\overline{\boldsymbol{A}}^t = \boldsymbol{A}^* = -\boldsymbol{A}$ Obviously, any symmetric matrix is also Hermitian (but not vice versa), and any skew-symmetric matrix is also skew-Hermitian (but not vice versa). It can also be shown that a Hermitian matrix has a symmetric real part and a skew-symmetric imaginary part, and a skrew-Hermitian matrix has a skew-symmetric real part and a symmetric imaginary part. Any real matrix can be written as the sum of a symmetric and a skew-symmetric matrix:

$$\boldsymbol{A} = \tfrac{1}{2}\left(\boldsymbol{A} + \boldsymbol{A}^t\right) + \tfrac{1}{2}\left(\boldsymbol{A} - \boldsymbol{A}^t\right) \tag{3.50}$$

and any complex matrix can be written as the sum of a Hermitian and a skew-Hermitian matrix:

$$\boldsymbol{A} = \tfrac{1}{2}\left(\boldsymbol{A} + \boldsymbol{A}^*\right) + \tfrac{1}{2}\left(\boldsymbol{A} - \boldsymbol{A}^*\right) \tag{3.51}$$

3.11 Work and Energy

When the structure is given a small (virtual) displacement [31][43] the corresponding virtual work due to external forces is

$$\delta W = \delta \boldsymbol{x}^t \boldsymbol{f} \tag{3.52}$$

Term	Default Units
$\boldsymbol{f}_t$	lb_f
$\boldsymbol{f}_r$	lb_f-in
$\hat{\boldsymbol{f}}_t$	lb_f-in
$\hat{\boldsymbol{f}}_r$	lb_f-in-rad
$\boldsymbol{x}_t$	in
$\boldsymbol{x}_r$	$radians(rad)$
$\boldsymbol{q}$	dimensionless
t subscript refers to translational degrees-of-freedom r subscript refers to rotational degrees-of-freedom	

Table 3.3: Vector Units

Term	Default Internal†Units
q	lb_f/in^2
ω	rad/sec
M	dimensionless
p	dimensionless
k	dimensionless
g	dimensionless
ρ	$lb_f sec^2/in^4$
s	rad/sec
d	dimensionless
σ	rad/sec
a	in/sec
V_t	in/sec
†internal units are used for computations	

Table 3.4: Parameter Units

Over a given period of time the work done is

$$W = \int_{t_1}^{t_2} \dot{\boldsymbol{x}}^t \boldsymbol{f}(t)\, dt \tag{3.53}$$

D'Alembert's principle [31][43] allows us to write an expression for the virtual work in the frequency domain as

$$\delta W = \delta \boldsymbol{q}^t \boldsymbol{D} \boldsymbol{q} \tag{3.54}$$

and over one cycle of oscillation

$$W = \int_0^{\frac{2\pi}{\omega}} \mathrm{Re}\left(s\, \boldsymbol{q}^t e^{st}\right) \mathrm{Re}\left(\boldsymbol{D}\boldsymbol{q}\, e^{st}\right)\, dt \tag{3.55}$$

so for undamped harmonic motion (i.e. neutral stability or flutter), the work done on the structure in one cycle of oscillation is

$$W = \sum_{k=1}^{n} \sum_{l=1}^{n} W_{kl} \tag{3.56}$$

where, using 3.17

$$W = \int_0^{\frac{2\pi}{\omega}} \mathrm{Re}\left(s\,\boldsymbol{q}^t e^{st}\right) \mathrm{Re}\left(\boldsymbol{D}\boldsymbol{q}\,e^{st}\right)\,dt \tag{3.57}$$

The dynamic matrix can be written as the sum of a Hermitian and a skew-Hermitian matrix (§3.51):

$$\begin{aligned} \boldsymbol{D} &= \boldsymbol{R} + \boldsymbol{S} \\ \boldsymbol{R} &= \tfrac{1}{2}\left(\boldsymbol{D} + \boldsymbol{D}^*\right) \\ \boldsymbol{S} &= \tfrac{1}{2}\left(\boldsymbol{D} - \boldsymbol{D}^*\right) \end{aligned} \tag{3.58}$$

which leads to

$$\begin{aligned} W &= -\omega \int_0^{\frac{2\pi}{\omega}} \sum_{k=1}^{n}\sum_{l=1}^{n} \Big[|q_k||q_l| R_{kl} \sin\left(\omega t + \phi_k\right) \cos\left(\omega t + \phi_l\right) \\ &\qquad\qquad - |q_k||q_l| S_{kl} \sin\left(\omega t + \phi_k\right) \sin\left(\omega t + \phi_l\right) \Big]\, dt \\ &= -\pi \sum_{k=1}^{n}\sum_{l=1}^{n} |q_k||q_l| \Big[R_{kl} \sin\left(\phi_k - \phi_l\right) - S_{kl} \cos\left(\phi_k - \phi_l\right) \Big] \\ &= -\pi\, \mathrm{Im}\left(\boldsymbol{q}^* \boldsymbol{D} \boldsymbol{q}\right) \\ &= i\pi \boldsymbol{q}^* \boldsymbol{S} \boldsymbol{q} \end{aligned} \tag{3.59}$$

That is, the work done on the system is only dependent upon the skew-Hermitian part of the dynamic matrix; mass and stiffness terms do not contribute, nor do gyroscopic terms, leaving only the skew-Hermitian part of the unsteady aerodynamic, viscous and structural-damping, and control-law matrices. The total work done on the structure at flutter is zero because $\boldsymbol{D}\boldsymbol{q} = \boldsymbol{0}$; however, useful information can still be gained from equation 3.57 if we only sum on k:

$$\boldsymbol{W}_l = i\pi q_l \boldsymbol{S}^* \boldsymbol{q} \tag{3.60}$$

which gives the work done on the structure by the l^{th} generalized-coordinate.

Chapter 4

Creating and Manipulating Matrices

There are several ways to create new matrices and manipulate existing matrices in FLAPS, including matrix algebra, extracting elements of matrices, merging matrices, and generating specialized matrices like gyroscopic and force matrices.

4.1 Matrix Algebra

Most of the operations defined on matrices are available in FLAPS using the alge command: matrix addition, subtraction, inversion, pseudo-inverse, transpose, inverse-transpose, real part, imaginary part, and conjugation. Matrices may be real or complex; operations involving both datatypes result in a complex matrix. For example if `Q` is an existing rectangular matrix the statement

```
alge { shouldbezero = (Q(T)*Q)(-1)*(Q(t)*Q) - Q(-1)*Q }
```

creates a new matrix `shouldbezero` which is square and contains all zeros.

Matrices may be multiplied or divided by real or complex scalars or scalar expressions; scalar expressions may include addition, subtraction, multiplication, division, exponentials, logarithms, square-root, powers, trigonometric functions (sin, cos, tan), inverse trigonometric functions (arcsin, arccos, arctan). For example,

```
alge { k = (1/sqrt(1 + i0.03))*AEKHH }
```

multiplies the real matrix by a complex scalar (1 over the square-root of 1+i0.03), producing a complex matrix (k). As another example, to evaluate the expression

$$\bar{\mathbf{M}} = \frac{1}{\sqrt[3]{2}}\mathbf{T}^t\mathbf{M}^{-1}\mathbf{T} \tag{4.1}$$

you could use

```
alge { Mbar = T(t)*M(-1)*T/pow(2,1/3) }
```

where `M` and `T` are existing matrices.

Select elements of matrices may be multiplied or divided by scalar expressions. For example

```
alge { k = (1/sqrt(1 + i0.03))*AEKHH{rows=(3 to 6), cols=(3 to 6) }
```

scales only a 3 by 3 diagonal block.

If a matrix is a function of one or more parameters, the matrix can be evaluated at specified values by following the matrix name with the parameter names and values enclosed in curly braces; for example

```
alge { g = gaf{rfi=0, mach=0.7, rfr=0} }
```

creates a new matrix (g) which is parameterized matrix gaf evaluated at the three parameters it is a function of.

Matrices that are functions of parameters may be treated in one of two ways: by evaluating the matrix at given values of the parameters, resulting in a constant matrix, or by operating on the parameterized matrix, resulting in a new parameterized matrix. For example,

```
param { gaf = genforce }
alge { igaf = gaf(-1) }
```

will interpolate all `genforce` matrices and create parameterized matrix `gaf`, then invert it and create parameterized matrix `igaf`; equivalently you could write

```
alge { igenforce = genforce(-1) }
param { igaf = igenforce }
```

which will invert each matrix named `genforce` with any attributes (§2.3) creating matrices named `igenforce` with the same sets of attributes, then interpolate these.

Matrix dimensions must be *conforming*: addition and subtraction require matrices with the same dimensions while multiplication requires the number of columns in the first matrix equal the number of rows in the second.

4.2 Extracting Elements

Rows, columns, or diagonals of existing matrices may be extracted to create new matrices using the extract command (§10.9). Parameterized matrices are treated in the same two ways as in the alge command: the extraction is done either on the matrix evaluated at specified values of the parameters, creating a constant matrix, or directly on the parameterized matrix, creating a new parameterized matrix.

4.3 Merging Matrices

Rudimentary matrix merging can be done in FLAPS with the merge command. Two or more matrices may be merged as rows or columns of a new matrix.

4.4 Gyroscopic Matrices

Gyroscopic forces arise from the motion of spinning flexible or rigid bodies; here we consider only rigid bodies whose spin axis passes through a structural node and whose orientation is determined by the rotational degrees of freedom at the node. The rate of change in orientation of the spin vector gives rise to accelerations which in turn produce moments acting on the rotational degrees of freedom of the node. This is analogous to the situation with translational degrees of freedom where according to Newton's second law the force is the rate of change of momentum

$$\boldsymbol{f} = \frac{d}{dt}(m\dot{\boldsymbol{x}}) \tag{4.2}$$

which for constant mass reduces to the more familiar vector equation

$$\boldsymbol{f} = m\ddot{\boldsymbol{x}} \tag{4.3}$$

For a rigid body spinning at a constant rate the analogous equation is

$$\boldsymbol{f} = \dot{\boldsymbol{x}} \times \boldsymbol{h} = -\boldsymbol{h} \times \dot{\boldsymbol{x}} \tag{4.4}$$

where $\boldsymbol{f}$ is a vector of moments, $\boldsymbol{x}$ is a vector of the rotational displacements of the node, $\boldsymbol{h}$ is the angular momentum vector of the part, and $\times$ is the vector cross-product. A gyroscopic matrix can be formed by writing the cross product in matrix form:

$$\boldsymbol{f} = -\begin{bmatrix} 0 & -h_3 & h_2 \\ h_3 & 0 & -h_1 \\ -h_2 & h_1 & 0 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} \tag{4.5}$$

Assembling all such 3 by 3 matrices yields the nodal gyroscopic matrix shown in equation 3.1 which then can be reduced as in equation 3.11.

The FLAPS gyro command (§10.11) is used to create gyroscopic matrices in either nodal or modal degrees of freedom for one or more spinning rigid bodies.

4.5 Force Vectors

Force vectors for use in the fresp or tresp commands may be created with the FLAPS force command.

Chapter 5

Parameters

Parameters play a central role in FLAPS commands, whether for studying the effects of parameters on vibration, stability or response, or simply specifying conditions for various analyses. There are a number of pre-defined (standard) parameters available in any FLAPS processor such as velocity and frequency; new parameters may be defined in most FLAPS processors using a common format described in this chapter. All parameters are *persistent*: they are available in subsequent processors, and changes to any parameter (standard or user-defined) take effect in all subsequent processors.

5.1 Parameter Format

All parameters in FLAPS are defined (and displayed) using a a common syntax referred to throughout this manual as a *parameter-defn*

name(*description*)[*min:max*]<*conversion*> = *equation*

The components of a parameter are

name Parameters are always referred to by name. There is no limit on the length of the name, but it may not contain spaces and probably should not contain commas.

(description) arbitrarily long description of the parameter, enclosed in parentheses. The description is never used to identify the parameter, only in printed output and plot files.

[min:max] Minimum and maximum acceptable values for the parameter in external units. Two floating-point numbers separated by a colon and enclosed in square brackets. If either number is missing the default is assumed.

<conversion> determines the units used within FLAPS for computations, the units

used for presentation in printout or plotfiles, and the conversion factor between the two.

equation The right-hand-side can be a simple value, multiple values enclosed in parentheses, or an equation. Standard parameters have multiple built-in equations (§5.5.1) which are chosen by FLAPS commands to give a consistent set of equations. User-defined equations (§5.4) may consist of matrix elements, parameter names, scalars, pre-defined constants (§2.5), +-*/, square root (sqrt), exponential (exp) and power (pow).

All the items except name are optional; however some contexts may require more than just a name. For example, it may be necessary to include a value or equation on the right-hand-side. See §5.8 for examples of valid parameter definitions.

The parameter name is case sensitive: `vtas`, `Vtas`, and `VTAS` are all different parameters. This implies that any parameter names in the right-hand-side equation are also case sensitive. The description and conversion units are only used for presentation purposes and are not case sensitive.

5.2 Parameter Units

Parameter units are determined by the <**conversion**> portion of a parameter definition, which in general consists of a *factor* a string describing the external (or *presentation*) units followed by a slash (/), followed by a string describing the internal (or *computational*) units. The factor is a floating-point number which when multiplied by a number in internal units yields a number in external units. The general format of a conversion is

<*factor* (*external units*)/(*internal units*)>

The strings describing the internal and external units should be enclosed in parentheses if the string contains a slash. These units descriptions are used in printout and plot files for information only; no magic here.

All computations are done in internal units, and parameter values are converted to external units by multiplying by the conversion factor for printed output and plotfiles.

Some examples of valid conversion factors:

```
<144 psf/psi>
<0.159154943 Hz/(rad/sec)>
<20736 (sl/ft^3)/(lbf-sec^2/in^4)>
```

Standard FLAPS constants (§2.5) may be used as conversion factors, with the added convenience that default units descriptions will be used. For example the default definition of frequency is

```
freq(Frequency)[.01:50]<HZPRS> = 0
```

which uses the built-in conversion HZPRS (Hz per radian/second).

5.2.1 Changing Units

The units associated with either the built-in standard parameters (§5.5) or user-defined parameters can be changed by redefining the parameter (anything but the name that is). External units can be changed to anything, but the internal units must remain compatible with the other parameters used in an analysis. For example, to change the external units for vtas from the default (knots) to furlongs/fortnight:

```
vtas(VelTAS)[0:3e+6]<152.727272 (furlong/fortnight)/(in/sec)>
```

Note that the limits, being in external units, have also been adjusted to give approximately 0 to 1000 knots, and the internal units have not been changed.

5.3 Parameter State

Active parameters are the independent variables in an analysis; for example, in a stability analysis velocity, frequency and growth rate might be active, while in a frequency-response analysis the frequency is active.

Fixed parameters have a value assigned by the user, usually in the options to the module. For example, including the option mach=0.6 in a stab run sets the Mach number to 0.6.

All other parameters are derived from the active and fixed parameters. The user is responsible for ensuring that the combination of fixed and active parameters is such that there is an equation for each derived parameter. For example, in a stability analysis if vtas, freq, and growth are active, then alt must be fixed in order to use the equation for mach $M = V_t/a(z)$. Currently available equations for the standard parameters are listed in section 5.5.1.

5.4 Parameter Equations

To the right of the equals sign in an equation definition is the parameter equation. A parameter equation can be as simple as a value (which makes the parameter state *fixed*), multiple values separated by commas and enclosed in parentheses, for example

```
gain = (1,2,3,4)
```

(which makes the parameter state *multiple-valued fixed*), or it can be a fairly arbitrary function of scalars, other parameters, builtin functions, and matrix elements, making the parameter state *derived*.

All of the pre-defined standard parameters (§5.5) have builtin equations, most have more than one. When an FLAPS processor is run with some of the standard parameters declared fixed, independent (active) or derived, the builtin equations are searched for compatibility with the declared parameters. It is an error if no compatible equation can be found.

User-defined equations may contain scalars, operators, other parameters, builtin functions, or matrix elements as detailed in sections 5.4.1-5.4.6. As usual, if an equation contains commas the equation must be quoted to protect the commas.

Equations, as all elements of parameter definitions, are *persistent*: after a parameter is given an equation, that equation will be used in subsequent processors until given another equation.

The ability to define parameter equations makes it possible to do nonlinear stability analyses using the *describing function* technique (appendix G)[30][45].

Parameter equations are made up of other parameters, matrix elements, scalars, operators, and functions, as described below.

5.4.1 Scalars

Scalars can be either real (e.g. 2.0 or 4e-5) or complex, as in (4.1 + i5.3) (§2.4). A complex scalar makes the resulting equation, and it's parameter complex.

Any of the pre-defined constants (§2.5) may be used; for example

```
sc = G*sin(PI/2)
```

5.4.2 Operators

The usual arithmetic operators can be used: * (multiply), / (divide), + (add) and - (subtract).

5.4.3 Other Parameters

Parameters are included in the equation by giving the parameter name. When the equation is evaluated the current values of any parameters are used, so they may in turn be functions of other parameters.

Derivatives of parameters may be used in equations by enclosing parameter names in d(); for example

```
dg = d(growth)/d(vtas)
```

defines a parameter (dg) as the derivative of growth with respect to vtas. Because there is in general no equation relating growth to vtas, the only place this has meaning is in the stab command where growth and vtas are related by the curve being traced. In this case dg gives the tangent to the growth-vtas curve at each solution point.

It is important to note that **the value of a parameter used in an equation is always in external units**. The reason is that if you specify a value for a parameter, the value is always assumed to be in external units; but because a value is merely the simplest form of equation, then the result of evaluating an equation must also be in external units.

5.4.4 Built-in Standard Functions

Several functions may be used in equations, including the standard trigonometric functions, logarithmic and exponential, and a few describing-functions which can be used to approximate nonlinearities. In the following table **x** and **y** are real or complex scalars or scalar expressions, with the exception of **min** and **max** which only take real scalars or scalar expressions.

abs(x)	Absolute value of x
acos(x)	arc cosine of x
asin(x)	arc sine of x
atan(x)	arc tangent of x
cos(x)	cosine of x
exp(x)	exponential of x: e^x.
log(x)	natural logarithm of x
log10(x)	base 10 logarithm of x
max(x,y)	maximum of x and y; x and y must be real
min(x,y)	minimum of x and y; x and y must be real
pow(x,y)	x raised to the y power: x^y.
sign(x)	sign of x: +1 if x is positive or -1 if negative
sin(x)	sine of x
sqrt(x)	Square root of x
tan(x)	tangent of x
tanh(x)	hyperbolic tangent of x. Tanh is sometimes used as an alternative to the sign(x) function because it does not have the discontinuity at zero that sign has, making it more suitable for time-integration.

5.4.5 Built-in Special Functions

A few functions are available to aid defining nonlinearities; these are all prefixed with ax:: (see appendix G):

ax::gap(fp, gc) given the size of a gap and the current value of a generalized-coordinate, returns a number between 0 and 1 which when multiplied by a stiffness value gives the equivalent stiffness accounting for the gap.

ax::gapdf(fp, gc) given the size of a gap (fp) and the current magnitude of a generalized-coordinate (gc), returns a number between 0 and 1 which when multiplied by a stiffness value gives the equivalent stiffness accounting for the gap using the describing-function technique.

ax::ffnasdf(h, gc) given the value of a static displacement (h) and the current magnitude of a generalized-coordinate (gc), returns a number between 0 and 1 which when multiplied by a stiffness value gives the equivalent stiffness for a failed mechanism.

ax::ffwpdf(fp, gc) given the size of a gap (fp) and the current magnitude of a generalized-coordinate (gc), returns a number between 0 and 1 which when multiplied by a stiffness value gives the equivalent stiffness accounting for the gap using the describing-function technique.

5.4.6 Matrix Elements

Matrix elements may be included in an equation by specifying the matrix id (§2.3) followed by the row and column enclosed in square brackets, for example `"gaf[3,5]"`. The matrix can be a function of parameters; it will be evaluated prior to using the element value. As usual, if the matrix id contains commas it must be enclosed in single or double quotes. For example

```
el = "AEMHH,MASSID=OEW"[6,6]
```

5.4.7 Evaluation

When the value of a parameter is needed, if it has an equation, the equation is evaluated; if the equation contains references to other parameters, those parameters are evaluated first. Likewise if matrix elements are included in the equation those matrices are evaluated and the relevant elements extracted and used in the equation. In either case the current value is used so the parameter or matrix element can themselves be variable.

5.4.8 Examples

Some examples of parameter definitions with equations:

```
eler = "2.0*elel*gaf[65,65]"
k = 378*freq2*freq2
dp = "sqrt(5.65e-5*vcas)/(pow(mach,0.02)*17.34)"
vc = exp(2*PI)*eler*DPR
```

Other examples are the equations available for the pre-defined standard parameters (§5.5.1).

5.5 Standard Parameters

In the standard presentation format (§5.1) the predefined parameters are

```
alt(Altitude)[-10000:250000]<ft> = 0

cdpress(Calib Dyn Press)[0:13000]<144 psf/psi> = 0

dpress(Dynamic Pressure)[0:13000]<144 psf/psi> = 0

freq(Frequency)[.01:50]<HZPRS> = 0

growth(Growth Rate) = 0

mach(Mach Number)[0:1] = 0

rfi(Reduced Frequency)[0:100] = 0

rfr(Real Part of p) = 0

rho(Std Atm Density)[0:1]<20736 (sl/ft^3)/(lbf-sec^2/in^4)> = 0

s(Characteristic Exponent) = 0

sdamp(Structural Damping) = 0

sigma(Real(Char. Exp)) = 0

spress(Static Pressure)[0:13000]<144 psf/psi> = 0

temp(Static Temperature)[0:660]<1 (deg K)/(deg K)> = 0

tpress(Stagnation Pressure)[0:13000]<144 psf/psi> = 0

vcas(Velocity (CAS))[0:1000]<KPIPS> = 0

veas(Velocity (EAS))[0:1000]<KPIPS> = 0

vsound(Sonic Velocity)[0:1000]<KPIPS> = 0

vtas(Velocity (TAS))[0:1000]<KPIPS> = 0
```

where HZPRS and KPIPS are among a number of predefined conversions (§2.5) usable anywhere you define a parameter.

Table 5.1 lists all the predefined parameters with their mathematical symbol and default units.

Name	Symbol	Description	Default Units	
			internal	external
alt	z	altitude	ft	ft
cdpress	q_c	calib. dynamic pressure	lb_f/in^2	lb_f/ft^2
dpress	q	dynamic pressure	lb_f/in^2	lb_f/ft^2
freq	ω	frequency	rad/sec	Hz
growth	γ	growth rate	-	-
mach	M	Mach number	-	-
p	p	complex reduced frequency	-	-
rfi	k	imaginary part of p	-	-
rfr	g	real part of p	-	-
rho	ρ	fluid density	$lb_f sec^2/in^4$	$slug/ft^3$
s	s	characteristic exponent	-	-
sdamp	d	structural damping coeff	-	-
sigma	σ	real part of s	rad/sec	Hz
spin	Ω	rotation rate	rad/sec	rad/sec
spress	P	static pressure	lb_f/in^2	lb_f/ft^2
temp	T	static temperature	$\deg K$	$\deg K$
tpress	P_t	total pressure	lb_f/in^2	lb_f/ft^2
vcas	V_c	calibrated airspeed	in/sec	$knots$
veas	V_e	equivalent airspeed	in/sec	$knots$
vsound	a	sonic velocity	in/sec	$knots$
vtas	V_t	true airspeed	in/sec	$knots$
- dimensionless				

Table 5.1: Standard Parameters

5.5.1 Standard Parameter Equations

Each of the standard parameters has one or more equations for calculating its value in terms of the other standard parameters when it is derived. Here we list the currently available equations; which equation is used depends on which parameters are active or fixed. Normally the user specifies which parameters are active or fixed and the program decides which equation to use.

Some parameters have default equations taken from the U.S. Standard Atmosphere as described in [7]. Properties of the standard atmosphere can be explored using the atmos option to the apex command (§10.6).

alt (Altitude)

reverse lookup on rho in tables of standard atmosphere properties

$$z = z(\rho) \tag{5.1}$$

dpress (Dynamic Pressure)

$$q = \frac{1}{2}\rho V_t^2 \tag{5.2}$$

$$q = \frac{1}{2}\rho(0) V_e^2 \tag{5.3}$$

freq (Frequency)

$$\omega = kV_t \tag{5.4}$$

$$\omega = 2\frac{\sigma}{\gamma} \tag{5.5}$$

growth (Growth Rate)

$$\gamma = 2\frac{\sigma}{\omega} \tag{5.6}$$

mach (Mach Number)

$$M = \frac{V_t}{a} \tag{5.7}$$

$$M = \sqrt{5\left\{\left[\left[\frac{V_c^2}{8.9710^8}+1\right]^{3.5}-1\right]^{.285714}-1\right\}} \tag{5.8}$$

p (Complex Reduced Frequency)

$$\begin{aligned} p &= \frac{s}{V_t} \\ &= g + ik \end{aligned} \tag{5.9}$$

rfi (Reduced Frequency)

$$k = \frac{\omega}{V_t} \tag{5.10}$$

rho (Density)

table lookup in standard atmosphere tables.

$$\rho = \rho(z) \tag{5.11}$$

s (Characteristic Exponent)

$$s = \sigma + i\omega \tag{5.12}$$

sigma (Real Part of s)

$$\sigma = \frac{1}{2}\gamma\omega \tag{5.13}$$

$$\sigma = Re(s) \tag{5.14}$$

spress (Static Pressure)

Also known as **ambient** or **freestream** pressure. Table lookup in standard atmosphere tables:

$$P = P(z) \tag{5.15}$$

tpress (Total Pressure)

Also known as **stagnation** pressure. Sum of dynamic and static pressure:

$$P_t = P + q \tag{5.16}$$

vcas (Calibrated Airspeed)

$$V_c = 12\sqrt{\frac{\gamma P(0)}{\rho(0)}\frac{2}{\gamma - 1}\left[\left[\frac{q}{P(0)} + 1\right]^{\frac{\gamma-1}{\gamma}} - 1\right]} \tag{5.17}$$

veas (Equivalent Airspeed)

$$V_e = V_t\sqrt{\frac{\rho(z)}{\rho(0)}} \tag{5.18}$$

$$V_e = \sqrt{\frac{2q}{\rho(0)}} \tag{5.19}$$

vsound (Sonic Velocity)

table lookup in standard atmosphere tables.

$$a = a(z) \tag{5.20}$$

vtas (True Airspeed)

$$V_t = \frac{M}{a(z)} \tag{5.21}$$

$$V_t = \frac{\omega}{k} \tag{5.22}$$

$$V_t = \sqrt{\frac{2q}{\rho(z)}} \tag{5.23}$$

$$V_t = V_e \sqrt{\frac{\rho(0)}{\rho(z)}} \tag{5.24}$$

5.6 Defining New Parameters

In addition to the standard parameters available automatically, new parameters can be defined and used just like the standard parameters. There a several places in FLAPS where new parameters can be defined:

- in the alge processor, when a new matrix is created, it may have new parameters associated with it
- in the param processor, when a new matrix is created, it may have new parameters associated with it
- parameter definitions with equations defined in terms of existing parameters can be included in stab, fresp, or tresp.

The syntax used to define new parameters is the same as that used for standard parameters (§5.1). These user-defined parameters can be defined with arbitrary equations in terms of existing standard and user-defined parameters.

New parameters are created in the param processor where they are associated with parameterized matrices, although they are also created anywhere a parameter is expected. For example in the stab command the option

```
stab { ..., pratio(Press Ratio) = "pow(1.0 + 0.2*mach*mach, 3.5)", ... }
```

will create a new parameter called pratio and define it as the ratio of total pressure to freestream pressure, using a basic equation of compressible flow. This new parameter could then be included in the plot file output by stab, where it will have the plot title "Press Ratio". Note the equation was enclosed in quotes to protect the comma from being interpreted as an option-separator.

Parameters in FLAPS are *persistent* in the sense that once defined they can be referenced by name in subsequent commands; the parameter's description, limits, conversion, and value are preserved until reset.

5.7 Output Transformation Parameters

Dynamic stability and response problems are usually done in terms of generalized coordinates, commonly represented by vibration modes. Often it is necessary to post-process these generalized coordinates, for example to visualize motion of the represented structure, or to compute strain energy. A special type of parameter, known as an *output transformation* or *ot* parameter is useful in these situations. The difference between an ot parameter and an ordinary parameter is that an ot parameter is a function of the generalized coordinates (and possibly other parameters). The function might be as simple as extracting one element of the generalized-coordinate vector or multiplying the vector by a row of the modes matrix associated with the generalized coordinates to get motion at a particular nodal degree-of-freedom.

An ot parameter is defined the same as any other parameter (§5.1) except that the equation contains a special parameter, the generalized coordinates, denoted by its name gc. For example, to define a parameters which are the velocity and acceleration at a particular nodal degree-of-freedom you could use

```
stab { ...,
   n7tzvel(Node 7 TZ Vel) = "s*n7tz*gc"
   n7tzacc(Node 7 TZ Accel) = "s*s*n7tz*gc"
   ... }
```

where n7tz is the name of a matrix with one row which is the row of the modes matrix corresponding to the tz freedom of node 7, possibly created with the extract command:

```
extract { n7tz = "modes,set=11"{row=7/tz} }
```

For convenience two more vectors are available which are the velocity and acceleration corresponding to gc, called gcdot and gcdotdot. With these vectors the above example could be written as

```
stab { ...,
   n7tzvel(Node 7 TZ Vel) = "n7tz*gcdot"
   n7tzacc(Node 7 TZ Accel) = "n7tz*gcdotdot"
   ... }
```

In a time-domain analysis in tresp the generalized-coordinates, velocities and accelerations have the same names, gc, gcdot, and gcdotdot, but in this case they are real vectors. The previous example in the time-domain would be

```
tresp { ...,
   n7tzvel(Node 7 TZ Vel) = "n7tz*gcdot"
   n7tzacc(Node 7 TZ Accel) = "n7tz*gcdotdot"
   ... }
```

5.8 Examples

A bewildering number of possibilities exist for defining parameters and re-defining existing parameters. Here we can only give a limited number of examples; hopefully these will inspire you to try others.

It is important to remember that parameters using in equations are evaluated in their **external** units, so to re-define reduced frequency it is necessary to convert frequency and velocity to computational (internal) units:

```
rfi(My Reduced Freq) = "(freq/HZPRS)*2/(vtas/KPIPS)"
```

which uses the built-in conversion factors HZPRS (Hz per radian/second) and KPIPS (knots per inch/second) (§2.5).

Multiple values can be specified for a parameter by enclosing the values in parentheses, just as you would any multi-valued right-hand-side. For example to specify that dynamic pressure is to have three constant values in a stab run you would include

```
..., dpress=(2,3,4.1), ...
```

but you could also use

```
..., dpress<6894.75 (N/m^2)/psi>=(2,3,4.1), ...
```

to change the units at the same time.

More examples of valid parameter definitions:

```
Ail(Aileron Freq)[0:10]<HZPRS> = 12345
fuel(Percent Fuel)[0:100]
gain = 1.3E-2
s = (3.14 + i6.28)
n7acc <.259007e-2 g/(in/s**2)> = -.2d-3
dpress = rho*pow(vtas,2)/2
```

For more examples of valid parameter definitions see section 5.5.

Chapter 6

Parameterizing Matrices

The process of creating a matrix which is a function of one or more parameters, known as *parameterization*, or *p14n* for short, is the role of the FLAPS param command. Matrices can be created which are functions of standard parameters (§5.1) or user-defined parameters (§5.6). These matrices may be used like ordinary matrices in other FLAPS commands, or they may be used in parameter studies in stab, fresp, or tresp. Most everywhere in FLAPS you can specify particular values for a parameterized matrix using the syntax

"mid"{param1=value1, param2=value2, ...}

that is, give the matrix id (name and attributes, §2.3) enclosed in quotes if it contains commas, followed by the parameter values enclosed in curly braces. For example,

```
print { Stif{ail = 10.3, elev=30.2} }
```

will print the matrix `Stif` with the parameter `ail` set to 10.3 and the parameter `elev` set to 30.2. If the matrix id contains commas it must be enclosed in quotes, for example

```
print { "AEMHH,MASSID=35"{center = 0, ob=50} }
```

will print the matrix `AEMHH,MASSID=35` with `center` set to 0, and `ob` set to 50.

The same syntax is used for associating new or existing parameters with a matrix as for specifying the parameter values when a matrix is to be evaluated: the matrix id followed by parameters enclosed in curly braces. The difference is that when associating parameters with a matrix it is necessary to specify how they are to be associated, whether they are to replace or scale certain elements, or if they represent branch mode frequencies. This is done with options enclosed in curly braces following the parameter definition.

Several types of parameterization are available:

- Matrices may be interpolated or approximated with respect to any number of parameters using splines of any order.
- Unsteady aerodynamic matrices may be approximated as a function of complex reduced frequency using a least-squares fit to a rational polynomial.
- Control-law matrices may be created using data from Matlab describing the `A, B, C and D` matrices from a Simulink control-law simulation (§C).
- User-written subroutines may be associated with a matrix to create arbitrary parameterizations of the matrix. Each time the matrix is evaluated the subroutine is called, where the matrix is created or modified.
- Individual elements of matrices may be made functions of parameters by either replacing or multiplying it by a parameter, which in turn may be defined in terms of other parameters using a parameter equation (§5.1). If the matrix is a stiffness matrix the parameter may represent a branch mode frequency (§H.2.2) or structural damping.

6.1 Interpolation and Approximation

Both interpolation and approximation are used in FLAPS to parameterize matrices and it is important to understand the difference between the two.

Interpolation forces a function (for example a polynomial) to pass through a given set of data. The resulting function exactly matches the data at least at the interpolation points; in between these points the accuracy is dependent on the type of functions used to interpolate.

Approximation, on the other hand does not require the function to pass through the data so in general the resulting function matches the data nowhere. It may however provide a better overall fit to the data in the sense that the largest error over the range of approximation may be less than with interpolation. Another advantage to approximation is that the resulting function may be smoother which may have important consequences for algorithms which depend on smoothness like the continuation method used in the FLAPS `stab` processor.

Figure 6.1 shows an example of the difference between approximated and interpolated data and why an approximation can be more accurate and smoother. It also illustrates a major drawback of approximation: there are an infinite number of approximations to a set of data depending on just how smooth the function is forced to be and this must be decided on a case-by-case basis.

6.1.1 Splines

Splines are a set of polynomials (piecewise polynomials) that are continuous at data points known as knots or breakpoints. The term comes from the draftsman's tool used to draw smooth curves. In fact the most commonly used polynomial spline, a natural

cubic spline, exactly models the draftsman's tool where the breakpoints correspond to the points where the draftsman places the weights (known as ducks). The term natural spline refers to a spline with zero second derivative at the ends, modeling the zero moment at the endpoints of the draftsman's spline. Generalization of mathematical splines are possible: fifth order polynomials (quintic splines) are sometimes used, various end and knot conditions are possible.

Splines can be used to either interpolate or approximate data. An interpolating spline is forced to pass through all the data points, sometimes resulting in a lack of smoothness in the curve. A smoothing spline is an approximation which relaxes the requirement to pass through the data points in order to give a smoother curve. Figure 6.1 shows a set of data points interpolated with a cubic spline and approximated with cubic splines using various levels of smoothing. The smoothing option in the param processor uses values ranging from 0 to infinity where 0 gives interpolation and the approximation approaches a straight line (linear least-squares approximation) as the smoothing parameter approaches infinity.

6.1.2 Interpolation

Matrices can be interpolated with respect to any number of parameters. Parameters may be defined in the param option list or (by default) taken from the attributes of the matrix. For example, unsteady aero matrices from BAP have the name `AEQHH` with attributes `rfi` and `mach`. Including the name `AEQHH` in the param option list causes param to interpolate all `AEQHH` matrices with respect to `rfi`, and if the `AEQHH` matrices were created at multiple values of `mach`, the interpolation with be with respect to the two parameters `rfi` and `mach`. Flutter analyses done with such a two-way interpolation will be more accurate than with just `rfi`-interpolated matrices.

Likewise, unsteady aerodynamic matrices created with the ATLAS/Flaps doublet-lattice processor have the name `genforce` with attributes `rfi`, `mach`, and `rfr`, so even more accurate flutter solutions are possible using a three-way interpolation. See demonstration problem stab3.ax for a comparison of different interpolation and approximation techniques.

Similarly a set of mass matrices can be interpolated based on a parameter that varies between each mass matrix. Given several mass matrices that correspond to the cg position of a control surface balance weight, a set of interpolation coefficients can be created with the param command.

There are times when it is necessary to interpolate matrices which are known to be discontinuous in slope with respect to the interpolation parameter. For example, on an airplane with multiple fuel tanks terms of the mass matrix will have slope discontinuities with respect to the fuel loading parameter at the point where one tank is emptied and fuel is taken from another tank.

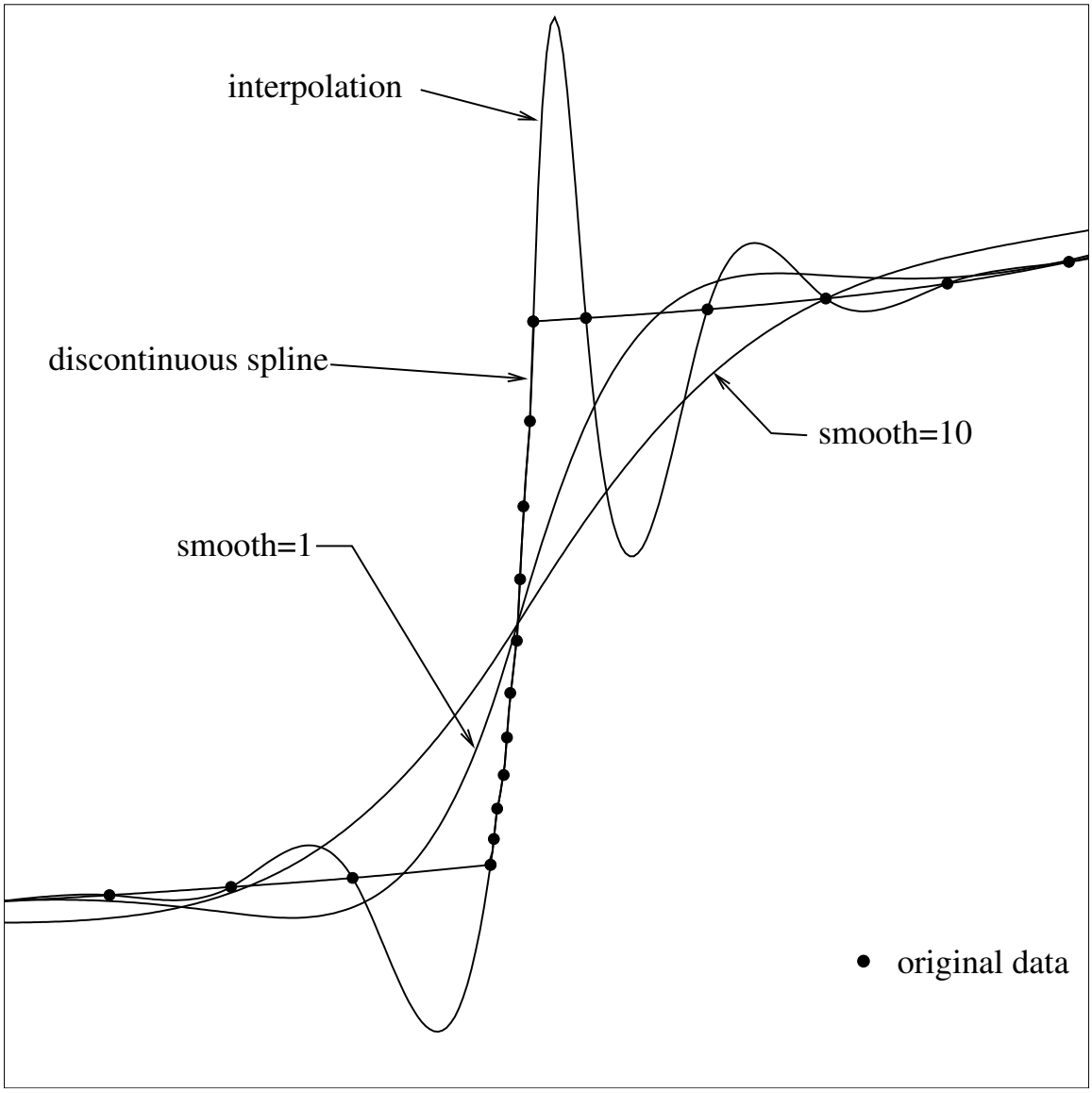


Figure 6.1: Interpolated and Approximated Data

6.1.3 Approximation

Interpolation results in a function which matches the original data exactly and approximates the data in-between the original data points; approximation on the other hand matches the original data nowhere in general, but may give an overall better approximation. A very important aspect of either interpolation or approximation is the type of function used. Interpolation in the param processor uses cubic-spline functions by default, so-called because they match the interpolation you would get with a drafter's spline. Interpolation constrains the function more than approximation so it is necessary to use functions that are less continuous; splines are continuous only up to the second derivative. With approximation you can use functions that are everywhere continuous or analytic over the complex plane. This is necessary for some applications, for example when integrating equations of motion which include unsteady aerodynamics (chap. **??**).

The param command approximates with splines when the smooth option is included, as discussed above. In addition the param command does one very specific type of approximation that is analytic: a least-squares fit of a rational polynomial due to Roger[42]. This type of approximation, also known as a *rational-function approximation (RFA)* or an *s-plane approximation*, is only available for unsteady aerodynamic matrices. Even though it is often referred to as an s-plane approximation, the least-squares fit is done with respect to complex reduced frequency. Appendix E contains details of this approximation.

The resulting approximation is

$$\boldsymbol{Q} \approx \boldsymbol{R}_0 + p\boldsymbol{R}_1 + p^2\boldsymbol{R}_2 + \sum_{i=1}^{m} \frac{p}{p+\beta_i}\boldsymbol{R}_{i+2} \tag{6.1}$$

where $p = \frac{s}{V_t}$ is the complex reduced-frequency (§5.5.1).

6.2 ABCD Control-Laws

Controls equations developed in Matlab/Simulink [23] can be exported from Matlab in the form of four matrices: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$. These matrices are exported to an ASCII file which can be processed by the param command to create a control-law matrix which may be used in stab or fresp. Appendix C details the way these state-space matrices are combined with the characteristic equations of motion (eqn. 3.41), and appendix D gives the format of the file containing the ABCD matrices.

In addition to the Matlab file the user must provide two matrices which represent input to and output from the control law: a `psi` matrix (represented by Ψ in appendix C) and either a `E` matrix or a `KE` matrix (represented by $\boldsymbol{E}$ and $\boldsymbol{KE}$ in appendix C)

The `psi` matrix relates the structural generalized-coordinates to physical displacements at the control-law sensors. It has a row for each control-law input and the same number of columns as generalized-coordinates, and is usually created from rows of the modes matrix associated with the generalized mass and stiffness matrices (see eqn. 3.12).

The E or KE matrix relates the control-law outputs to generalized forces on the structure. The number of rows is the same as the number of structural generalized-coordinates, and the number of columns is the number of control-law outputs. Typically the KE matrix consists of columns of the generalized stiffness matrix, possibly with sign and units changes to account for flight controls conventions. The E matrix is multiplied internally by the stiffness matrix to produce the KE matrix; it is important to understand the difference between using a KE matrix and using a E matrix. The two approaches are equivalent if the stiffness matrix K is constant; it is when elements of the stiffness matrix that contribute to the product KE are not constant that it is necessary to use the E matrix instead of the KE matrix. For example, if an output is to a control surface represented by generalized-coordinate 150, the KE matrix would be column 150 of the stiffness matrix; but if that coordinate has been parameterized so that it is a function of a branch mode frequency parameter and you want to do a parameter variation on that parameter, then you must supply an E matrix consisting of column 150 of the identity matrix, which will then multiply the stiffness matrix, which contains the correct value of the frequency parameter.

6.2.1 A-Matrix Interpolation

The ***A*** matrix is, in general a function of dynamic pressure, or an equivalent parameter such as altitude for a constant-Mach analysis or true airspeed for a constant-altitude analysis. This relationship is passed in ABCD files from Matlab/Simulink by a set of cubic-spline interpolation coefficients. A typical element of an ***A*** matrix is shown in figure 6.2

In theory, cubic splines satisfy the continuity requirements of the FLAPS flutter solution technique (§**??**) but it is possible to create splines that are so nearly discontinuous in the first derivative that they can cause tracking problems. For example, figure 6.3 shows the derivative with respect to altitude of this element and a point where a mode failed to track.

Smoothing the spline interpolations [27] can cure tracking problems and speed up the tracking process. The param processor includes an option, smooth which modifies the spline from Matlab/Simulink to make it smoother. This option specifies a smoothing factor which determines how much the spline is straightened out: zero results in the original spline, while the curve approaches a straight line as the smoothing factor approaches infinity. Values between 1 and 10 result in reasonably smooth curves while not changing the original values too much. Figure 6.4 shows the results of two values of the smoothing factor, and figure 6.5 shows the derivative of the smoothed splines.

Smoothing the splines has the additional benefit of speeding up the mode-tracking process. The algorithm used in stab to determine stepsize uses estimates of the curvature of the curve being traced; kinks in the curve cause the algorithm to take small steps, slowing the process down. Figure 6.6 shows how a mode with a slope discontinuity is smoothed out using various levels of smoothing. None of the smoothing levels make noticeable changes in the curve without zooming in.

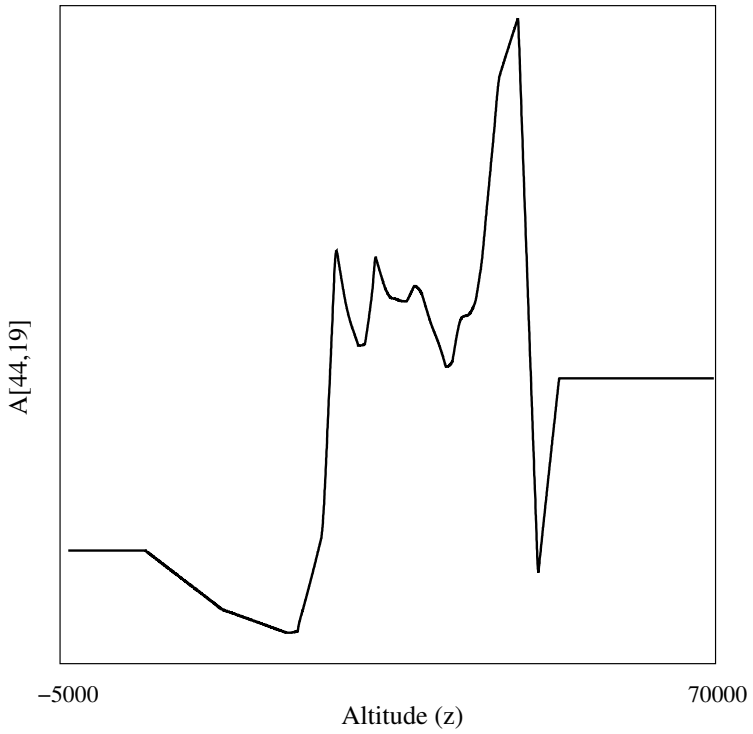


Figure 6.2: Typical Element Variation with Altitude

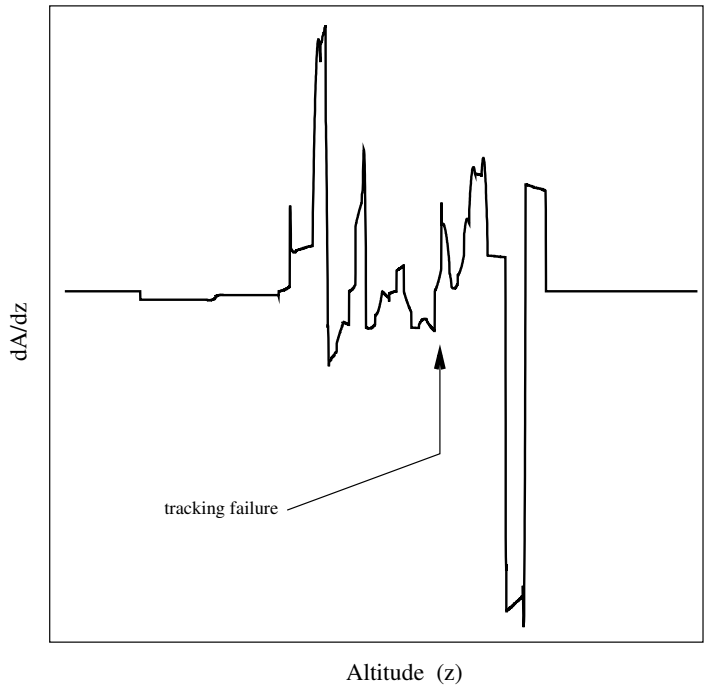


Figure 6.3: Spline Derivative

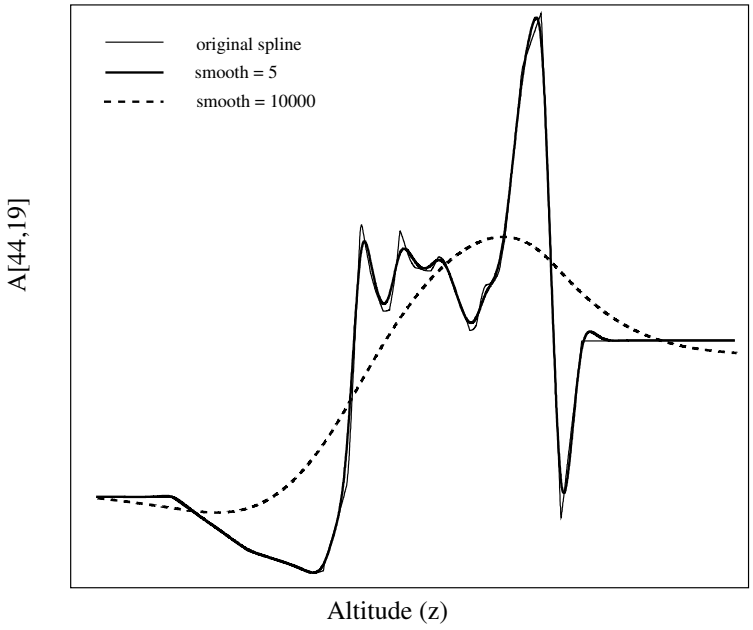


Figure 6.4: Smoothed Spline

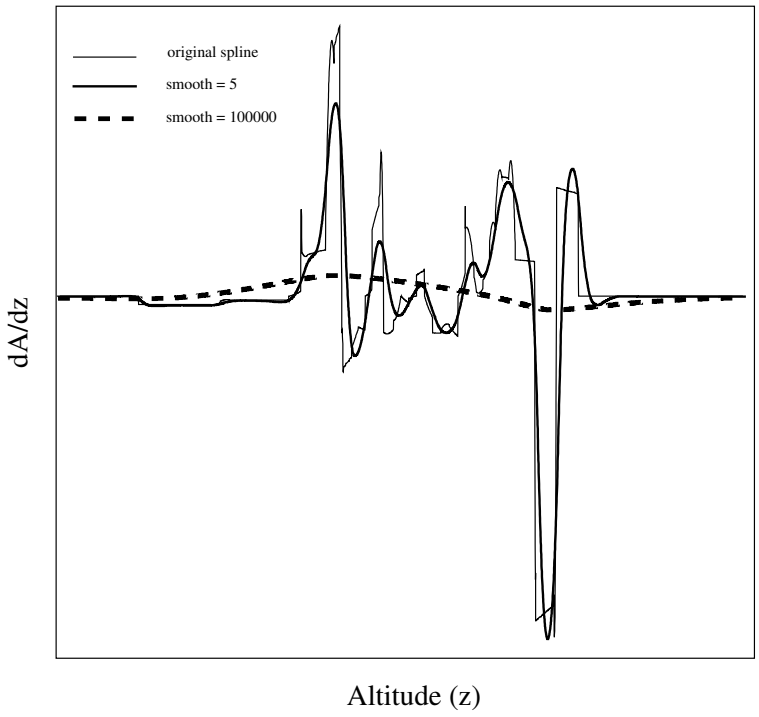


Figure 6.5: Smoothed Spline Derivative

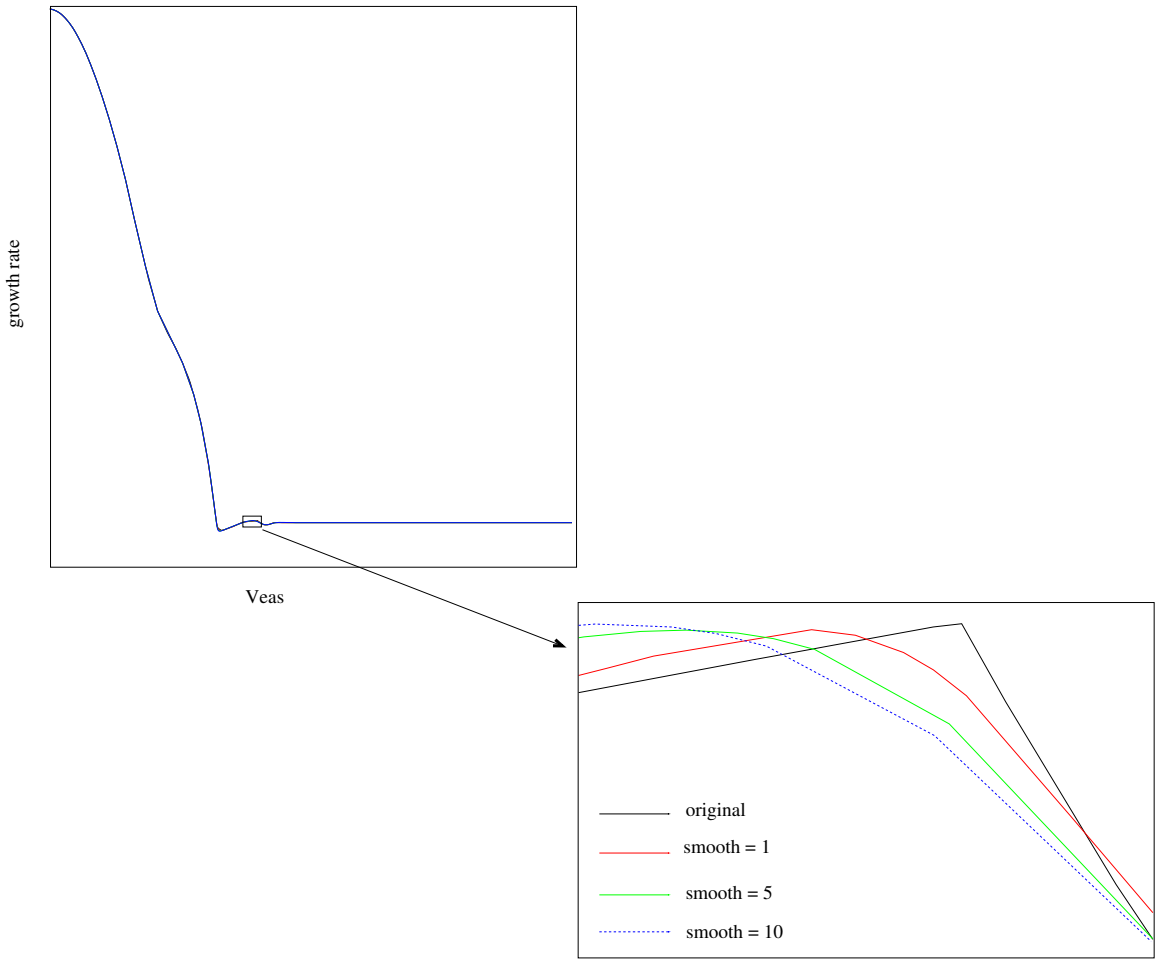


Figure 6.6: Aeroelastic Mode with Smoothing

6.3 Matrix Elements

Individual matrix elements may be parameterized by giving them an equation, using the same syntax, scalars, operators, other parameters, and functions as the equations used to define parameters (§5.4). The parameterized matrix element is specified by a pair of square brackets enclosing the row and column numbers, for example to set row 3, column 5 to 4.0:

```
[3,5] = 4.0
```

Giving a matrix element an equation creates a new parameter which is then known to all subsequent FLAPS commands by a name formed from the matrix name and element indices; for example, giving the above equation to a matrix called `AEKHH` creates a new parameter named `AEKHH[3,5]`.

One additional syntactic element is used to refer to the original value of the matrix element. A pair of square brackets, used anywhere in an equation, will be replaced by the value of the matrix element. For example to scale row 3, column 5 by 4.0:

```
[3,5] = 4.0*[]
```

What follows are some examples of how this capability can be used.

6.3.1 Branch Mode Frequencies

Branch Modes (appendix H) is a dynamic substructuring technique in which a structure is split into components, then assembled in such a way that the individual components exhibit the *branch mode property* (§H.2.2): there is mass coupling but no stiffness coupling between components; as a consequence, scaling diagonal elements is the same as scaling the cantilevered natural frequencies of the component. This property is often used to set diagonal elements of stiffness matrices to *branch mode frequencies*: diagonal elements of the resulting generalized stiffness matrix are directly proportional to the square of the natural frequencies of the component if it were cantilevered at the interface between it and neighboring components.

To replace an element of a stiffness matrix with a function of a branch mode frequency, you could use the relation

$$\omega_{ij} = \sqrt{\frac{K_{jj}}{M_{jj}}} \tag{6.2}$$

For example the command

```
param {
   i = AEKHH
   freq77(G.C. 77 Freq)<HZPRS>
   [7,7] = "freq77*freq77*AEMHH[7,7]"
}
```

creates a new parameter (`freq77`) with a built-in conversion factor (`HZPRS`) and uses it in an equation which sets the (7,7) diagonal element of the stiffness matrix `AEKHH` to a value that gives a branch mode frequency of `freq77`. A more straightforward method is to use the bmfgc option in the param command (§10.15.3).

6.3.2 Structural Damping

There are two ways to include structural damping in an analysis: by specifying the value of d in equation 3.23 with the `sdamp` parameter (§5.5), giving the same structural damping coefficient to all degrees of freedom.

Alternatively each degree of freedom may be given a different structural damping coefficient by defining a structural damping parameterization (sdp14n) in which one or more degrees of freedom are given a structural damping coefficient specified by a new parameter; this type of parameterization is defined with the param command.

A few cautionary notes regarding structural damping:

- if `sdamp` and sdp14n are both used the stiffness matrix is first multiplied by the sdp14n and then by `sdamp` so some elements will be multiplied twice.
- with sdp14n entire rows and columns for each generalized coordinate are multiplied by $1 + id_k$; more precisely each element in the row or column is replaced by the real part of the element times $1 + id_k$. This means that coupling terms have structural damping applied once, but it also means that if multiple sdp14n are defined and there is coupling between the generalized coordinates in them, the coupling terms will have the structural damping of the last defined sdp14n.

6.3.3 Nonlinear Analyses

Equations describing matrix elements may be nonlinear functions of other parameters, built-in functions, and generalized-coordinates. If the equation contains (implicitly or explicitly) time or generalized coordinates, analyses with this matrix are nonlinear and generally require more work than linear equations.

For example, if generalized-coordinate 33 represents rotation of a control surface, and the control surface has 0.1 radian freeplay, the stiffness matrix can be parameterized in one of two ways depending upon whether it is to be used in frequency-domain or time-domain analyses.

In the time domain the stiffness element can be replaced with the exact function of displacement:

```
param {
   i=AEKHH, o=Stif
   fp(G.C. 33 Freeplay) = 0.1
   gc33(G.C. 33 Disp) = gc[33]
   [33,33] = []*gap(fp,gc33)
}
```

where `gap` is one of the built-in functions (§5.4.5) which takes two arguments, the gap size and the current value of the generalized coordinate. Two new parameters are defined to set the value of the gap (`fp`) and the current value of generalized-coordinate 33 (`gc33`), using the built-in function `gc` to get the value at any point in the solution process. It is necessary to use the built-in function `gap` because the equation for the scale factor it returns is too complicated for a single equation (the real problem is that the FLAPS equation capability does not allow logic statements such as `if`):

```
Real
gap(Real fp, Real gc) {
   Real factor;
   if (gc33 > gap) {
      factor = (gc33 - gap)/gc33;
   } else if (gc33 < -gap) {
      factor = (gc33 + gap)/gc33;
   } else {
      factor = 0.0;
   }
   return factor;
}
```

For analyses in the frequency domain it is necessary to approximate the nonlinearity using a *describing function* (appendix G):

```
param {
   i=AEKHH, o=Stif
   gap(G.C. 33 Freeplay) = 0.1
   gc33(G.C. 33 Magn) = 1
   [33,33] = "[]*((PI - 2*asin(gap/gc33) - sin(2*asin(gap/gc33)))/PI)"
}
```

for example, setting a stiffness term with

```
[3,5] = 4.0*[]
```

A few describing functions are available for approximating nonlinear behavior of individual generalized coordinates.

6.3.4 Nodal Matrix Elements

Sometimes it is desirable to modify a generalized matrix to reflect changes in the underlying nodal matrix without re-doing the matrix triple-product in equation 3.11. This can be done for a single nodal degree-of-freedom provided there is no coupling between it and any other nodal degrees-of-freedom.

Assuming the nodal matrix has the form

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{1j} \\ \boldsymbol{A}_{j1} & A_{jj} \end{bmatrix} \tag{6.3}$$

where the j^{th} degree-of-freedom is to be modified, and A_{jj}, $\boldsymbol{A}_{1j}$, and $\boldsymbol{A}_{j1}$ are the rows and columns to be modified. The modes matrix has the form

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_1 \\ \boldsymbol{\Phi}_j \end{bmatrix} \tag{6.4}$$

where $\boldsymbol{\Phi}_j$. is the row of the modes matrix corresponding to the element to be modified. The generalized matrix is

$$\begin{aligned} \bar{\boldsymbol{A}} &= \boldsymbol{\Phi}^t \boldsymbol{A} \boldsymbol{\Phi} \\ &= \boldsymbol{\Phi}_1^t \boldsymbol{A}_{11} \boldsymbol{\Phi}_1 + \boldsymbol{\Phi}_j^t A_{jj} \boldsymbol{\Phi}_j + \boldsymbol{\Phi}_1^t \boldsymbol{A}_{1j} \boldsymbol{\Phi}_j + \boldsymbol{\Phi}_j^t \boldsymbol{A}_{j1} \boldsymbol{\Phi}_1 \end{aligned} \tag{6.5}$$

which, because A_{jj} is just a scalar can be written for any value as

$$\bar{\boldsymbol{A}}(\gamma) = \bar{\boldsymbol{A}}^0 + \left(A_{jj}(\gamma) - A_{jj}^0 \right) \Phi_j^t \Phi_j \tag{6.6}$$

where $\bar{\boldsymbol{A}}^0$ is the original generalized matrix, A_{jj}^0 is the original value of the element to be modified, and the modified value $A_{jj}(\gamma)$ is a function of some parameter γ. Modifications to degree-of-freedom j in the nodal matrix can therefore be reflected in the generalized matrix simply by modifying A_{jj}, subtract the original A_{jj}, multiply it by the matrix $\boldsymbol{\Phi}_j^t \boldsymbol{\Phi}_j$, and add it to the generalized matrix.

For example, to represent freeplay in the j^{th} nodal degree-of-freedom with a describing function $c(\gamma)$,

$$\boldsymbol{K}(\gamma) = \bar{\boldsymbol{K}} + \left(c(\gamma) - 1 \right) K_{jj} \boldsymbol{\Phi}_j^t \boldsymbol{\Phi}_j \tag{6.7}$$

6.4 User-Subroutine Parameterization

The most general parameterization, a Fortran or C subroutine is used to create (or modify) a matrix. The subroutine can have any name but must have a very specific set of calling arguments:

```
subroutine anyname (nr, a)
```

where `nr` is the row dimension of the matrix, and `a` is the matrix itself. Moreover, the matrix must be declared `real` or `complex` as it is in the param command:

```
subroutine anyname (nr, a)
real a(nr,*)
```

or

```
subroutine anyname (nr, a)
complex a(nr,*)
```

Subroutines written in Fortran must conform to the Fortran77 standard even though more modern versions of Fortran are available; this is due to the fact that in the stab command it is necessary to take derivatives of all matrices. A technique called *automatic differentiation* (appendix K) is used to do this with no extra effort on the part of the user, but the current implementation uses a Fortran77-to-C++ translator (§K.2), hence the restriction to standard Fortran77.

In addition to the standard Fortran functions and subroutine calls, several FLAPS-specific subroutines are available (see appendix B for more detail).

`parval` any parameter, standard (§5.1) or user-defined (§5.6) are accessed in internal units using the `parval` function call:

```
subroutine anyname (nr, nc, a)
complex a(nr,nc)
complex s
s = cmplx(parval('sigma'), parval('freq'))
```

`getpar` similar to `parval` except it returns current parameter values in external units.

```
subroutine anyname (nr, nc, a)
complex a(nr,nc)
real vtas
vtas = getpar('vtas')
```

`setpar` sets the value of a parameter (real parameters only); the value is given in external units and is converted to internal units by `setpar`.

```
subroutine anyname (nr, nc, a)
complex a(nr,nc)
real vtas
vtas = 400
call setpar('vtas', vtas)
```

`matvij` Elements of matrices are available with the `matvij` subroutine; for example to get the (3,3) term of the mass matrix:

```
complex mij
call matvij('mass', 3, 3, mij)
```

`fetch` Entire FLAPS matrices can be read using the `fetch` subroutine, but it's a bit more complicated. The array is placed in memory in a location relative to an input array with an index returned in the fifth argument; for example

```
      real mat(1)
      character*8 dtype
      call fetch('ADIRU', m, n, mat, ip, dtype, iprint, irr)
c get the (4,5) term:
      i = 4
      j = 5
      a45 = mat(ip-1+i+m*(j-1))
```

Sometimes it is desirable to plot quantities that are computed within user-written subroutines. This is done by defining a parameter which will hold values in the FLAPS command which uses the subroutine; then every time the subroutine is called the current value is set by calling `setpar`, for example

```
c(5,5) = tf1*tf2*tf3
call setpar(cabs(c(5,5)), 'c55')
```

will set the value of a parameter called `c55` to the current absolute value of the `(5,5)` element of the matrix `c`. The parameter must be defined either in the param statement that creates the user-subroutine parameterized matrix, as in

```
param {..., c55(C diagonal 5)=0, ... }
```

or in the stab, fresp, or tresp statement that uses the parameterized matrix, for example

```
stab {..., c55(C diagonal 5)=0, ... }
```

When the matrix is evaluated, the nominal matrix (which may consist of all zeros) is passed to the subroutine where terms of the matrix are set or modified according to the current values of the parameters. Demo problems alge3.ax and stab4.ax create new matrices using a user-subroutine; stab2.ax and stab12.ax evaluate complex matrices representing controls equations, and stab6.ax modifies an existing matrix using a user-written subroutine.

Chapter 7

Flutter

Flutter equations are called **linear** if they are linear in the generalized coordinates; that is, if the dynamic matrix (eqn. 3.42) is not a function of the generalized coordinates. Even though these equations are nonlinear functions of system and user-defined parameters such as velocity, frequency, and growth-rate, they are called nonlinear only if the dynamic matrix is a function of the generalized coordinates.

The term **nonlinear** Flutter equations refers to equations where the dynamic matrix (usually just the structural stiffness or aerodynamic matrices) are functions of the generalized coordinates. In such cases the basic assumption of harmonic motion (eqn. 3.15) no longer holds and in general nonlinear problems must be solved in the time domain. Often, nonlinear equations can be solved in the frequency domain by approximating the nonlinearities under the assumption of harmonic motion; this technique, known as *describing functions* [30][45], replaces matrix terms which are nonlinear functions of the generalized coordinates with the equivalent function obtained by assuming harmonic motion. There is a huge computational advantage to solving flutter equations in the frequency domain: a solution in the frequency domain determines stability directly but in the time domain it is necessary to integrate the equations of motion until steady state is reached. Determining the exact conditions for neutral stability in the time domain is very time-consuming, whereas in the frequency domain it is relatively easy.

As discussed in section 3.3 the stability of a linear system is determined by the real part of the characteristic exponent $s = \sigma + i\omega$. Positive values indicate an unstable structure, negative values a stable one, and zero values indicate neutral-stability. The imaginary part determines the frequency of oscillation.

The FLAPS flut command solves the flutter equation in a continuous fashion between the limits of the parameters. Continuity of solutions is what distinguishes the solution technique from more traditional techniques [20] [5] [12]. When the flutter equation is solved in a continuous fashion it becomes possible to solve not only for neutral-stability points as the traditional techniques do, but to do *parameter variations*, studying the influence of various parameters. Continuity is important when solving the flutter equation because often it is very difficult to distinguish aeroelastic modes without some mathematical assurance that points on a plot are connected.

The flutter equation may contain user-defined parameters and arbitrary auxiliary equations typically used to model the effects of active controls. When used together with the param module, the user has great flexibility in defining matrices to be arbitrary functions of standard parameters (§5.5) or user-defined parameters (§5.6).

Often the interest in flutter equations is in solving for *neutral-stability*, the boundary between stability and instability where small oscillations in the structure neither grow nor decay with time. We use the term *growth rate* as a measure of the rate at which oscillations are growing. This term is preferred over the more traditional term *decay rate* because a positive value of growth rate indicates growing oscillations. The term *flutter crossing* or *flutter speed* refers to a velocity at which the growth rate has a specified value; neutral-stability is therefore a zero flutter crossing, and an "oh-three crossing" is where the growth rate equals 0.03.

Neutral-stability calculations are usually just a necessary first step leading to a number of *parameter variations* where, starting from a flutter crossing a parameter is varied, resulting in a *stability boundary*. The parameter varied can be any system parameter (§5.1) or user-defined parameters (§5.6). Two parameters can be varied to give contours of flutter speed; and multiple parameters can be varied to give a type of optimization which we call *continuation optimization*.

7.1 Flutter Equation

Linear stability of a structure is determined by computing roots of the homogeneous (no forcing function) characteristic equation (see chapter 3 and equation 3.41):

$$\boldsymbol{Dq} = \left[s^2\boldsymbol{M} + s\boldsymbol{G} + s\boldsymbol{V} + (1+id)\boldsymbol{K} - q\boldsymbol{Q}(p,M) + \boldsymbol{T}\right]\boldsymbol{q} = \boldsymbol{0} \tag{7.1}$$

A *root* or *eigenpair* of this equation is a pair $(s, \boldsymbol{q})$ which satisfies equation 7.1, and is more commonly referred to as an *aeroelastic mode*. In theory there are n aeroelastic modes for an n^{th}-order system of equations, but the interest is usually in low-frequency aeroelastic modes which tend to be represented more accurately.

Scalar quantities used in these equations are listed in table 5.1, reproduced in table 7.1. Matrices are listed in table 3.1. Not all quantities in table 7.1 appear explicitly in equation 7.1; for example, true airspeed appears only in the definitions of dynamic pressure, reduced frequency, and Mach number. Also not explicit are user-defined parameters; any of the matrices may be functions of standard and user-defined parameters (see chap. 6).

A solution to equation 7.1 requires the dynamic matrix is singular, meaning the determinant of the dynamic matrix is zero. The determinant is a complex number so we can choose two real variables (e.g. freq and sigma) for a system of two real equations (real and imaginary parts of the determinant) in two unknowns. Adding an third unknown allows us to trace a curve; adding a fourth unknown yields a surface, and so on. The unknowns, which we call *active parameters*, determine the type of solution. For example, the traditional *k method* [20] (also known as the *V-g method*) uses vtas, freq,

Name	Symbol	Description	Default Units	
			internal	external
alt	z	altitude	ft	ft
cdpress	q_c	calib. dynamic pressure	lb_f/in^2	lb_f/ft^2
dpress	q	dynamic pressure	lb_f/in^2	lb_f/ft^2
freq	ω	frequency	rad/sec	Hz
growth	γ	growth rate	-	-
mach	M	Mach number	-	-
p	p	complex reduced frequency	-	-
rfi	k	imaginary part of p	-	-
rfr	g	real part of p	-	-
rho	ρ	fluid density	$lb_f sec^2/in^4$	$slug/ft^3$
s	s	characteristic exponent	-	-
sdamp	d	structural damping coeff	-	-
sigma	σ	real part of s	rad/sec	Hz
spin	Ω	rotation rate	rad/sec	rad/sec
spress	P	static pressure	lb_f/in^2	lb_f/ft^2
temp	T	static temperature	$\deg K$	$\deg K$
tpress	P_t	total pressure	lb_f/in^2	lb_f/ft^2
vcas	V_c	calibrated airspeed	in/sec	$knots$
veas	V_e	equivalent airspeed	in/sec	$knots$
vsound	a	sonic velocity	in/sec	$knots$
vtas	V_t	true airspeed	in/sec	$knots$
- dimensionless				

Table 7.1: Flutter Equation Parameters

and sdamp; setting vtas, freq, and sigma active results in the so-called *p method* if the aerodynamic matrix is a function of real and imaginary reduced-frequency, or the *p-k method* if it is only a function of reduced-frequency.

Most of these formulations have associated solution techniques and there is often no effort to distinguish between the formulation and solution technique; for example, the V-g method casts the flutter equation into a complex generalized eigenvalue problem, for which there are stable and reliable solution techniques and the term V-g method usually refers to both the formulation and the eigenvalue solution. The solution technique presented here is capable of solving virtually any formulation of the flutter equation; for example you can solve the V-g form of the equations by using structural damping as an active parameter. The difference is that in flut the equations can include nonlinearities, control-laws, gyro, unsteady aerodynamics that are functions of Mach and complex reduced frequency, and can do parameter variations.

A few combinations of active and fixed parameters are listed in table 7.2; others are possible as long as they result in a consistent set of equations for evaluating all parameters (see §5.5.1 for available equations).

The flut command requires the user to declare three parameters to be active; in addition, some parameters must be declared *fixed* or *multiple-valued fixed* (§5.4) parameters. The remainder are *derived parameters*, evaluated using one of the equations in section 5.5.1. It is the user's responsibility to ensure that the chosen active and fixed parameters

result in a consistent set of parameter equations. Table 7.2 shows fixed parameters which result in consistent parameter equations for a few common situations.

Name(s)	Actives	Fixed	Description
cmcd p-k method p method	vtas freq growth	alt	Constant-Mach-constant-density p-k flutter solution Aero not a function of Mach
k method v-g method	vtas freq sdamp	alt growth	To simulate a traditional k-method flutter solution. See demo problem stab10.ax.
cmvd	alt or veas freq growth	mach	Constant-Mach-variable-density p-k flutter solution
parameter variation	alt, veas, or vtas freq `p1` †	growth	Parameter variation where `p1` is, e.g. fuel-loading, nacelle frequency, control-law gain

flutter speed contours	`p1`
	`p2`
	freq
		growth
		alt or vtas
			Usually done at multiple alt, veas, or vtas to represent a 3D surface of flutter speed as a function of two parameters

optimization	`p1`
	`p2`
	...
	freq
		growth	The resulting curves move in the direction of greatest increase and decrease of the parameter specified with the `optimize` option

† `p1, p2`, etc can be any parameter			

Table 7.2: Common Flutter Solution Parameters

7.1.1 V-g Formulation

A simplification of equation 7.1, known as the *V-g* formulation is obtained by neglecting viscous damping, gyroscopics, and control-systems, and assuming harmonic motion ($\sigma = 0$):

$$\left[-\omega^2 \boldsymbol{M} + (1 + id)\boldsymbol{K} - q\boldsymbol{Q}(k)\right] \boldsymbol{q} = \boldsymbol{0} \tag{7.2}$$

which can be written in the form of a generalized eigenvalue problem (§**??**):

$$\begin{aligned} \boldsymbol{A}\boldsymbol{q} &= \lambda \boldsymbol{B}\boldsymbol{q} \\ \boldsymbol{A} &= \boldsymbol{K} \\ \boldsymbol{B} &= \boldsymbol{M} + \frac{\rho}{2k^2}\boldsymbol{Q} \\ \lambda &= \tfrac{\omega^2}{1+id} = \lambda_r + i\lambda_i \end{aligned} \tag{7.3}$$

from which

$$\begin{aligned} d &= -\frac{\lambda_i}{\lambda_r} \\ \omega &= \sqrt{1 + d^2}\lambda_r \end{aligned} \tag{7.4}$$

Because there are robust methods for solving the generalized eigenvalue problem, the V-g formulation has been widely used for many years to solve for aeroelastic stability. Note that instead of solving for the characteristic exponent s this eigenvalue problem gives the structural damping necessary to result in undamped motion; traditionally it has been referred to as *added structural damping* because positive values are necessary to cause an unstable structure to move sinusoidally (neutrally-stable), and negative values to make a stable structure neutrally-stable - physically improbable but the values are close to the values of growth rate obtained from equation 7.1 (see eqn. 3.29).

A drawback with the V-g formulation is that it is not able to track the evolution of rigid-body modes with increasing speed; at any speed rigid-body modes result in zero eigenvalues of equation 7.3 which does not reflect the actual motion of an airplane. Rigid-body modes usually have aerodynamic forces on them which generally increase the frequency with increasing speed, a fact that is missing from the V-g formulation. The basic assumption in the V-g formulation is that it is possible to add structural damping (positive or negative) to a structure which will result in harmonic motion; this assumption breaks down with rigid-body modes because there is no structural deformation, hence structural damping has no influence. This is probably the reason the V-g formulation is able to predict the correct divergence speed (§3.5.3).

7.1.2 Divergence

Another simplification of equation 7.1 is to set the frequency to zero and search for conditions where σ is also zero. Like the flutter equation this condition is the boundary between stability ($\sigma < 0$) and instability ($\sigma > 0$); unlike the flutter equation this is a *static* condition and the speed where it occurs is known as the *divergence* speed. Equation 3.41 reduces to

$$[(1 + id)\boldsymbol{K} - q\boldsymbol{Q}(0, M) + \boldsymbol{T}(0, q)]\,\boldsymbol{q} = \boldsymbol{0} \tag{7.5}$$

which is a nonlinear eigenvalue problem (§**??**) with eigenvalues q and eigenvectors $\boldsymbol{q}$. Methods for solving this equation are in §7.7.

If the unsteady aerodynamic matrix is not a function of Mach number this equation can be solved by computing, for each in a series of m dynamic pressures spanning the flight conditions of interest, the ordinary eigenvalue problems

$$[(1 + id)\boldsymbol{K} - q_i\boldsymbol{Q}(0) + \boldsymbol{T}(0, q_i)]\,\boldsymbol{q} = \lambda\boldsymbol{I} \quad (i = 1, m) \tag{7.6}$$

for eigenpair $(\lambda, \boldsymbol{q})$ and watch for $\lambda = 0$.

Without controls equations and with an unsteady aerodynamic matrix that is not a function of Mach number the equation is further simplified to

$$[(1 + id)\boldsymbol{K} - q\boldsymbol{Q}(0)]\,\boldsymbol{q} = \boldsymbol{0} \tag{7.7}$$

which is a generalized eigenvalue problem (§**??**) for eigenpair $(q, \boldsymbol{q})$ and now it is simply a matter of picking the smallest (real) q.

7.1.3 Continuation Optimization

A limited type of optimization may be performed in flut by declaring four or more active parameters and declaring a parameter to be optimized with the optimize option. Because there are more than three active parameters, there are an infinite number of directions the solution curve can go; at each step we choose the direction which results in the greatest change in the optimize parameter. Demo problem stab8.ax shows an example of optimizing flutter speed with respect to nacelle frequencies.

7.2 Solution Technique

The method used to solve the flutter equation was developed at Boeing to solve the problem of tracking aeroelastic modes in a continuous fashion. Traditional flutter solution techniques ([12] [5]) required the user to join discrete solution points, often guessing which points connect. In addition, the new flutter solution technique is able to trace parameter variations using the same method as the basic flutter solution. More details can be found in [33].

Equation 7.1 is a system of algebraic equations linear in the generalized-coordinates and nonlinear in most other variables listed in table 7.1. From this set of n complex equations an equivalent set of $2n$ real equations can be derived by considering the real and imaginary parts of the *residual* vector $\boldsymbol{r} = \boldsymbol{D}\boldsymbol{q}$ as separate real variables. Because equation 7.1 is homogeneous in the generalized coordinates it is necessary to add a constraint to eliminate the trivial solution $\boldsymbol{q} = \boldsymbol{0}$. Here we add a normalization condition

$$\begin{aligned} \boldsymbol{q}^*\boldsymbol{q} &= 1 \\ \mathrm{Im}(u_k) &= 0 \end{aligned}$$

The resulting set of $2n + 2$ nonlinear real equations to be solved are

$$\boldsymbol{f}(\boldsymbol{x}) = \left\{ \begin{array}{c} \mathrm{Re}(r_1) \\ \mathrm{Im}(r_1) \\ \vdots \\ \mathrm{Re}(r_n) \\ \mathrm{Im}(r_n) \\ \boldsymbol{q}^*\boldsymbol{q} - 1 \\ \mathrm{Im}(u_k) \end{array} \right\} = \boldsymbol{0} \tag{7.8}$$

where Re and Im indicate the real and imaginary parts, respectively, $\boldsymbol{f}$ is a real vector of length $2n+2$, and $\boldsymbol{x}$ is a vector comprising active parameters from table 7.1 and the real and imaginary parts of the generalized-coordinate amplitudes ($\boldsymbol{q}$). The number of elements in $\boldsymbol{x}$ is $2n$ (real and imaginary parts of the generalized-coordinates) plus the number of active parameters. The number of active parameters determines the nature of the solution: 2 results in a point because $2n+2$ equations in $2n+2$ unknowns has a unique solution; 3 active parameters result in a curve, and 4 results in a surface. Table

7.2 shows a few combinations of active and fixed parameters. With the exception of optimization, all these combinations consist of 3 actives, so the solutions are curves.

Equation 7.8 with the dimension of $\boldsymbol{x}$ greater than $\boldsymbol{f}$ can be solved using a *continuation method*. [4]. A continuation method works much like an ordinary differential equation solver: starting from a known solution and using derivatives of the independent variables a new solution at new values of the independent variables is predicted, then the new solution is solved for using a correction scheme such as Newton's method. The stab module uses a modified version of a general-purpose package [40][38][39] called PITCON for solving continuation problems which have one more independent variable than dependent variables. In our case this code solves the problem where the number of unknowns in $\boldsymbol{x}$ is $2n + 3$; that is, three variables can be chosen in addition to the generalized-coordinates. This is the reason for the requirement discussed in section 7.1 to declare three active parameters. This is the basic technique for solving all types of problems in stab involving three variables.

7.2.1 Optimization Technique

With more than three active parameters in $\boldsymbol{x}$ there is additional freedom to choose the direction in which a curve is traced. Forcing the curve to proceed in the direction of greatest (positive or negative) change in some parameter results in a kind of optimization; because these solutions are obtained using the same continuation method used for all other types of flutter solutions, we refer to it as *continuation optimization*. Figure 7.1 shows an optimization curve where the optimization parameters are percent center and main fuel and the curve traces the path of greatest change in vtas.

7.3 Start Points

The technique used to track aeroelastic modes requires start points; finding start points is a common source of problems when using the FLAPS stab processor. In this section some of the problems associated with start points are explained, with some advice for mitigating these problems.

The process for obtaining start points depends on whether or not the analysis starts from previous solutions. Analyses that start from a previous solution are those which include the source option to specify the previous solution. Analyses that do not include the source option are usually (but not necessarily) neutral stability (or pk) solutions; parameter-variation (or pv) solutions usually (but not necessarily) include the source option. A common chain of analyses is a pk solution followed by one or more pv solutions using the pk solution as the source, although other scenarios are possible, such as a pv solution starting from another pv solution varying different parameters, or a pk solution starting from a pv solution. Combinations like these are possible because FLAPS treats all parameters the same: a pk solution is really just a pv solution with velocity, frequency, and damping as parameters.

When tracking from previous solutions, that is when using the source option, start

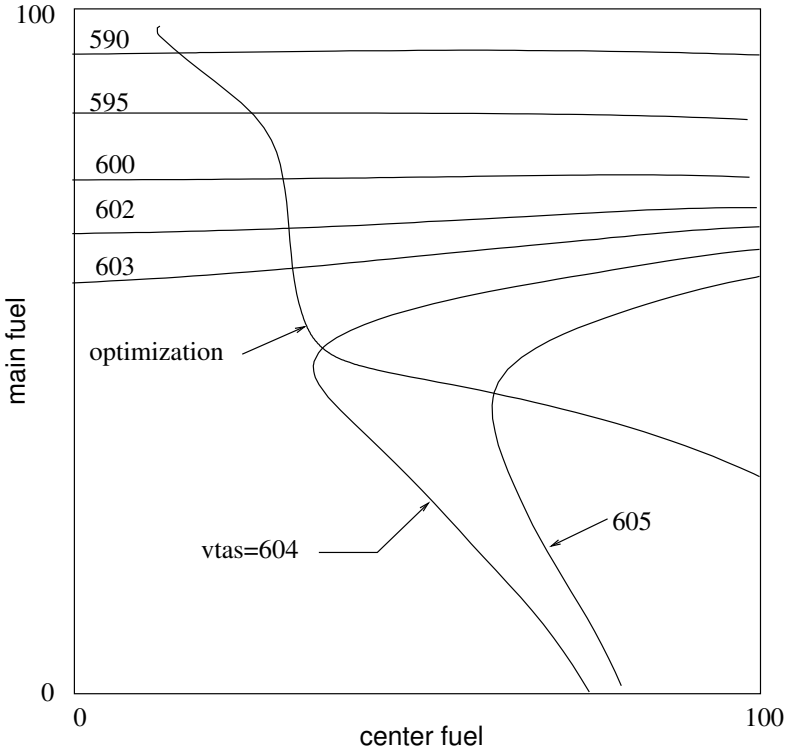


Figure 7.1: Fuel Optimization and Contours

points are obtained by straightforward interpolations of the previous solutions; this does not mean that the process is foolproof, as shown below in the discussion of *parameter cuts*.

Obtaining start points when the source option is not included can be much more difficult and less robust. The general solution technique for solutions which do not start from previous solutions involves a nonlinear eigenvalue problem (§**??**). Unfortunately there is no known method for guaranteeing we have found all eigenvalues within a region with nonlinear eigenvalue problems. The best we can do is narrow the frequency range where we look for start points (see the startregion option in section **??**).

Generally, analyses without control-law matrices (so-called *open-loop* problems) do not have a problem with start points because they are not highly nonlinear as *closed-loop* analyses often are. In fact open-loop problems are polynomial eigenvalue problems, quadratic in the characteristic exponent, for which robust solution techniques are available.

It is really only closed-loop analyses for which we have no reliable techniques for computing start points yet, though this is an active research topic.

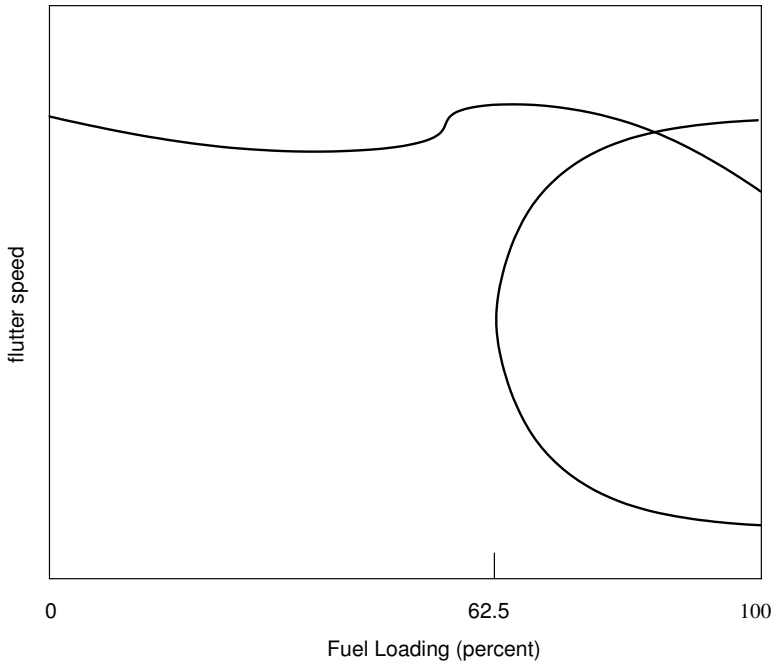


Figure 7.2: Fuel Variation

7.3.1 Parameter Variation Cuts

When the source option is included, finding start points is easy; the problem is that the start points (and the source analysis) may not be at values of the parameters which yield all parameter-variation curves of interest. Generally, the source analysis must be done at multiple values of the parameter (or parameters) which are to be varied in the parameter-variation analysis. Analyses done at multiple parameter values are known as *cuts* for reasons that should be clear with an example.

Figure 7.2 shows a typical parameter variation: the variation in flutter speed (where growth rate is zero) with percent fuel.

If the pk solution was done at a value of fuel between 0 and 62, the pk curve would be as in figure 7.3, with one place where growth rate is zero.

However, the C shaped curve does not show up in the pk solution at values of fuel between 0 and 62.5; above 62.5 the pk solution would look more like figure 7.4.

In order to get a start point for the C-shaped curve it is necessary to do a pk solution with fuel somewhere between 62.5 and 100. For this reason it is necessary to do pk solutions at a number of values of the parameter (fuel in this case). This is done by specifying multiple values of the parameter in the pk solution; for example

```
stab { ..., fuel=(0,10,40,80,100), ... }
```

Unfortunately, the best way to ensure that a parameter variation curve is not missed is

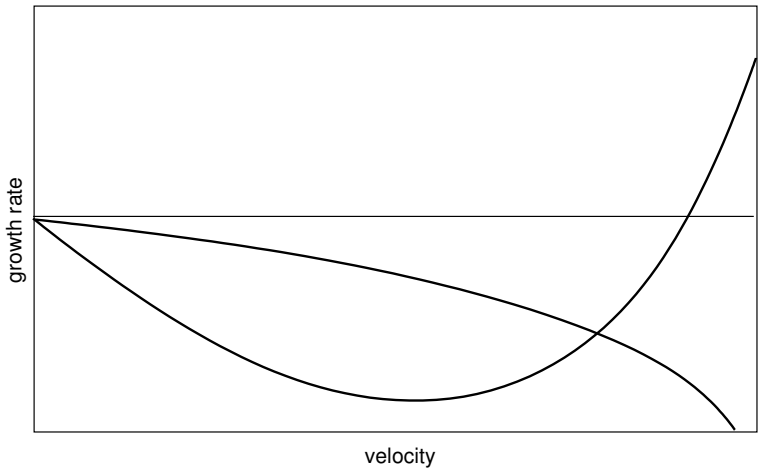


Figure 7.3: Flutter Solution at fuel=0

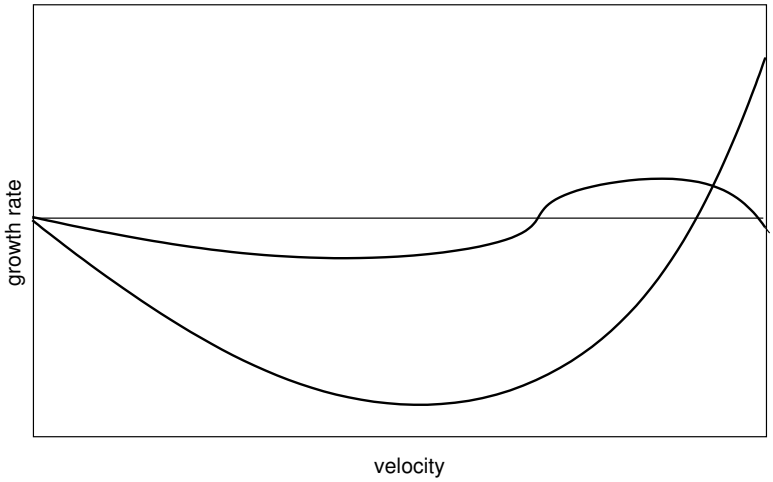


Figure 7.4: Flutter Solution at fuel=100

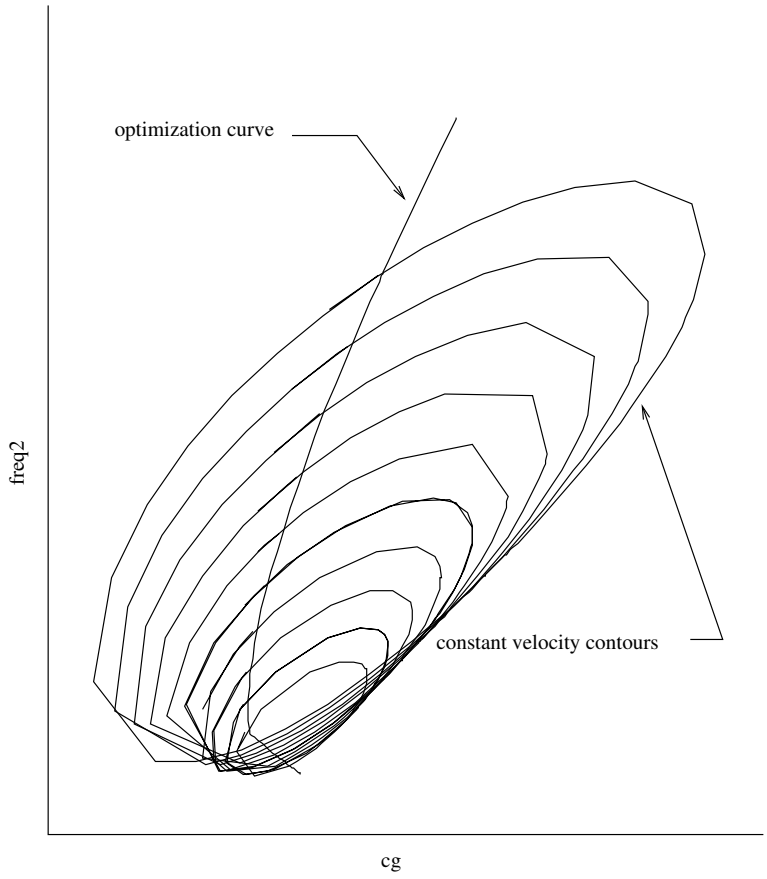


Figure 7.5: Contours From an Optimization Curve

to include enough parameter values ("cuts") in the pk solution; this can be expensive and time-consuming but it is the best method at this point.

7.3.2 Contours

Parameter variations involving two parameters result in parameter contours. Start points for contours can be obtained by doing parameter cuts as with single parameter variations, but a safer and more effective technique is to use the continuation optimization method (§7.2.1).

Figure 7.5 shows an example of the technique taken from demo problem stab1.ax where a curve of vtas optimized against cg and freq2 is used to get start points for a series of contours of cg vs freq2 at various values of vtas. Viewed in 3 dimensions, the contours would outline a cone-shaped surface of flutter speed (vtas) versus cg and freq2.

To see why optimization works better than a simple parameter variation to get start points for tracking contours, consider what happens if a single-parameter variation is

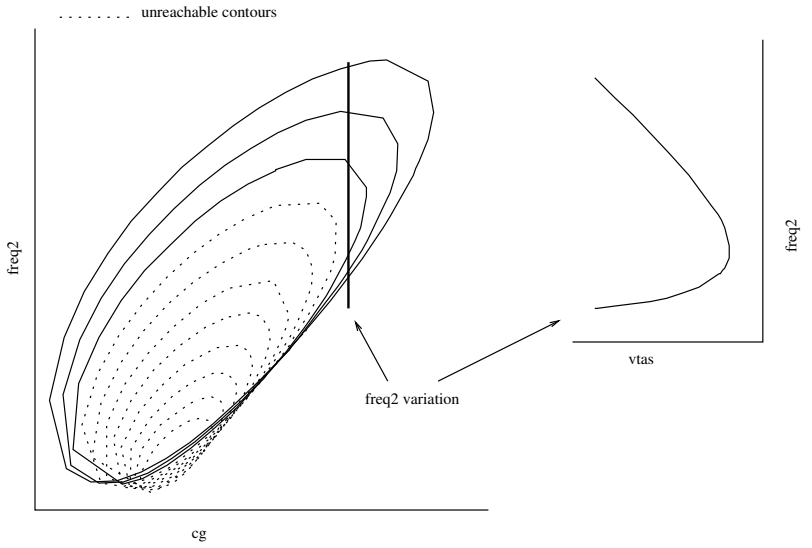


Figure 7.6: Contours From a freq2 Variation

used to get start points for the above example. Figure 7.6 shows a slice through the cone at a fixed value of cg: a freq2 parameter variation. Clearly the values of vtas in this parameter variation do not extend as far as the optimization curve; thus the resulting cone would be truncated.

7.4 Fluid Properties

Several of the variables in table 7.1 are properties of the fluid. This section describes some of the possibilities for controlling these properties.

7.4.1 Standard Atmosphere

By default, analyses in the stab module use the standard atmosphere as defined in [7]. In the standard atmosphere the sonic velocity, density, and static pressure are functions only of altitude; thus two types of analyses possible in the standard atmosphere are constant-altitude and variable-altitude.

In a constant-altitude analysis the altitude is specified by the user, thus fixing a, ρ, and P. If the unsteady aerodynamics are defined for a single Mach number (the *aero Mach number*), the true Mach number $M = V_t/a$ will not match the aero Mach number. In other words, a *constant-Mach constant-altitude* (*cmcd*) analysis assumes the aerodynamics are not a function of Mach number. If, on the other hand, the unsteady aero matrices are functions of Mach number, the aero Mach number **will** match the true Mach number (see demo problem stab3.ax). Unsteady aero matrices that are functions

of Mach number are created by parameterizing sets of matrices at more than one Mach number.

Variable-altitude analyses hold the Mach number constant and vary the altitude, effectively varying the density and sonic velocity, and therefore the true airspeed $V_t = Ma$ and equivalent airspeed $V_e = V_t\sqrt{\frac{\rho(z)}{\rho(0)}}$

7.4.2 Simulating a Wind Tunnel

In a wind tunnel the static temperature, static pressure, and Mach number are nearly constant; if it is assumed that the gas is a perfect gas, then the speed of sound is also constant. Setting these parameters to "fixed" and setting veas, freq, and growth to "active" in a stab run gives an approximation to conditions in a wind tunnel.

7.5 Unsteady Aerodynamics

The Doublet-Lattice Method (DLM) [2] is the most widely used method for generating matrices of unsteady aerodynamic forces in subsonic flow. The DLM will generate aerodynamic matrices at specified values of Mach number and reduced frequency k or, with modification, complex reduced frequency (p). For a flutter solution it is necessary to be able to evaluate the unsteady aerodynamic matrix at any value of Mach number and complex reduced frequency. Generating aero matrices is a relatively expensive operation so the usual technique is to generate unsteady aerodynamic matrices at specific values and estimate at intermediate values using interpolation or approximation. Sections 6.1.3 and 6.1.2 discuss how to do this in FLAPS. The current approximation method in FLAPS is not as accurate as interpolation methods, so interpolation is the preferred technique for flutter analyses.

The most general way to treat unsteady aerodynamic matrices is to generate a number of them at various values of complex reduced frequency and Mach number and interpolate with respect to the three (real) parameters, rfr, rfi, and mach. This requires many more generated matrices than if the interpolation is done with respect to rfi only, or rfi and mach. The decision depends largely on the type of flutter solution to be done. If the interest is in finding flutter points (where growth and rfr are zero) and tracing the flutter boundary with parameters it is probably not necessary to interpolate with respect to rfr. If the interest is more with *subcritical* (where oscillations decay) behavior, interpolating with respect to rfr would be more accurate. Similarly, interpolating with respect to mach gives more accurate results for solutions at a constant altitude, but is unnecessary for solutions at a constant Mach number.

There have been attempts to approximate unsteady aerodynamics for non-zero values of rfr. Two of these are available in the stab command using the nastran and gmethod options. Details of these two methods are in section 3.5.2.

7.6 Nonlinear Stability

If the matrices in the flutter equation (7.1) are functions of time or the generalized-coordinates the flutter equation is nonlinear and must be solved using time-integration. In certain cases generalized-coordinate nonlinearities can be approximated using the describing-function technique (appendix G). Nonlinear stiffnesses associated with branch mode generalized coordinates (appendix H) can be given a describing-function equation in the param processor. Branch mode generalized coordinates are characterized by mass coupling but a lack of stiffness coupling between it and other generalized coordinates. This makes it possible to replace the single stiffness term associated with that generalized coordinate with a describing function equation. The equation can be an arbitrary function of other parameters, built-in functions, or generalized-coordinate amplitudes (§5.4).

For example, if generalized-coordinate 33 represents rotation of a control surface, and the control surface has 0.1 radian freeplay, the stiffness matrix can be parameterized in one of two ways depending upon whether it is to be used in frequency-domain or time-domain analyses.

In the time domain the stiffness element can be replaced with the exact function of displacement:

```
param {
   i=AEKHH, o=Stif
   fp(G.C. 33 Freeplay) = 0.1
   gc33(G.C. 33 Disp) = gc[33]
   [33,33] = []*gap(fp,gc33)
}
```

where gap is one of the built-in functions (§5.4.5) which takes two arguments, the gap size and the current value of the generalized coordinate. Two new parameters are defined to set the value of the gap (fp) and the current value of generalized-coordinate 33 (gc33), using the built-in function gc to get the value at any point in the solution process. It is necessary to use the built-in function gap because the equation for the scale factor it returns is too complicated for a single equation (the real problem is that the FLAPS equation capability does not allow logic statements such as if):

```
gap(fp, gc) {
   if (gc33 > gap) {
      factor = (gc33 - gap)/gc33
   } else if (gc33 < -gap) {
      factor = (gc33 + gap)/gc33
   } else {
      factor = 0.0
   }
   return factor
}
```

and its rotational stiffness is specified with an equation that multiplies the nominal stiffness by a gap describing function:

```
param {
```

```
   i=AEKHH, o=Stif
   gap(G.C. 33 Freeplay) = 0.1
   gc33(G.C. 33 Magn) = 1
   [33,33] = "[]*((PI - 2*asin(gap/gc33) - sin(2*asin(gap/gc33)))/PI)"
}
```

where gap is a parameter defined and given an initial value of 0.1, and gc33 is another parameter which we will use in stab to force the generalized-coordinates to be normalized to specific values. These new parameters can given values in a neutral-stability analysis; the generalized-coordinate vector can be normalized to the current value of gc33 using the norm option:

```
stab { ...,
  gap = 0.02
  gc33 = 1
  norm{gc=33, val=gc33}
  ...
}
```

Equivalently, the built-in function gapdf (§5.4.5) can be used in the equation:

```
param {
   i=AEKHH, o=Stif
   gap(G.C. 33 Freeplay) = 0.1
   gc33(G.C. 33 Magn) = 1
   [33,33] = "[]*ax::gapdf(gap,gc33)"
}
```

The above stab command is for a particular value of displacement; usually the interest is in studying the variation in flutter speed with the magnitude of displacement of the nonlinear coordinate:

```
stab { ...,
  active=(gc33[0.02:2],freq,vtas)
  growth=0
  gap = 0.02
  ...
  norm{gc=33, value=gc33}
  pstz = pstzmat*gc
}
```

which tracks the variation in flutter speed (growth=0) with g.c. 33 amplitude and normalizes the generalized coordinates so that g.c. 33 has the value of gc33. This normalization is necessary to give the proper output transformation in parameter pstz.

7.7 Divergence

Another type of instability occurs when the stiffnesses are not enough to overcome the aerodynamic forces, resulting in potentially destructive deformations. This is a *static* phenomenon: no oscillations are involved, the frequency is zero, no inertia forces are

involved, and the mass is irrelevant. As with flutter the boundary between stability and instability is when $\sigma = 0$ and the divergence equation is (§7.1.2)

$$[(1 + id)\boldsymbol{K} - q\boldsymbol{Q}(0, M) + \boldsymbol{T}(0, q)]\,\boldsymbol{q} = \boldsymbol{0} \tag{7.9}$$

Without controls equations the divergence equation is (§7.1.2)

$$[\boldsymbol{K} - q\boldsymbol{Q}(0, M)]\,\boldsymbol{q} = \boldsymbol{0} \tag{7.10}$$

which is a generalized eigenvalue problem (§**??**) and the eigenvalues are the dynamic pressures where divergence occurs. The unsteady aerodynamic matrix is evaluated at zero reduced frequency; if it is also a function of Mach number the dynamic pressure and Mach number will not, in general be consistent. In this case or if there are controls equations a more complicated procedure is necessary. One technique is to consider the dynamic matrix as a function of dynamic pressure

$$\boldsymbol{D}(q) = [(1 + id)\boldsymbol{K} - q\boldsymbol{Q}(0, M) + \boldsymbol{T}(0, q)]\,\boldsymbol{q} = \boldsymbol{0} \tag{7.11}$$

and at each dynamic pressure in the range of interest compute the eigenvalues of $\boldsymbol{D}$. Speeds where the smallest eigenvalue is zero are divergence speeds. Both of these techniques are available in the FLAPS stab command.

7.8 Continuity

The solution technique used to track modes in stab was chosen because it (almost) guarantees continuous solution curves, making it especially useful for parameter variations. No algorithm is perfect, however, and the tracking algorithm in stab can fail in several ways:

- Start points. The most common failure in stab is in neutral-stability runs, where the start points are found by solving a nonlinear eigenvalue problem at small or zero dynamic pressure. Without controls equations the problem is much simpler and less prone to failure; with controls the problem can be highly nonlinear and difficult to guarantee that all start points have been found.
- Mode switching. Modes commonly cross other modes and in doing so if the values of all parameters are nearly the same (including the generalized-coordinates) it is conceivable that the modes could switch or both modes continue tracking the same mode. The algorithm in stab considers the generalized coordinates the same as ordinary parameters as it traces modes, making switching extremely unlikely.
- Mode reversal. It is possible when tracking a mode through a region where parameters are changing rapidly for the algorithm to reverse direction and continue tracking the mode back the way it came. In neutral-stability runs this would result in a message like

```
Finished tracking: max alt (250000) reached
```

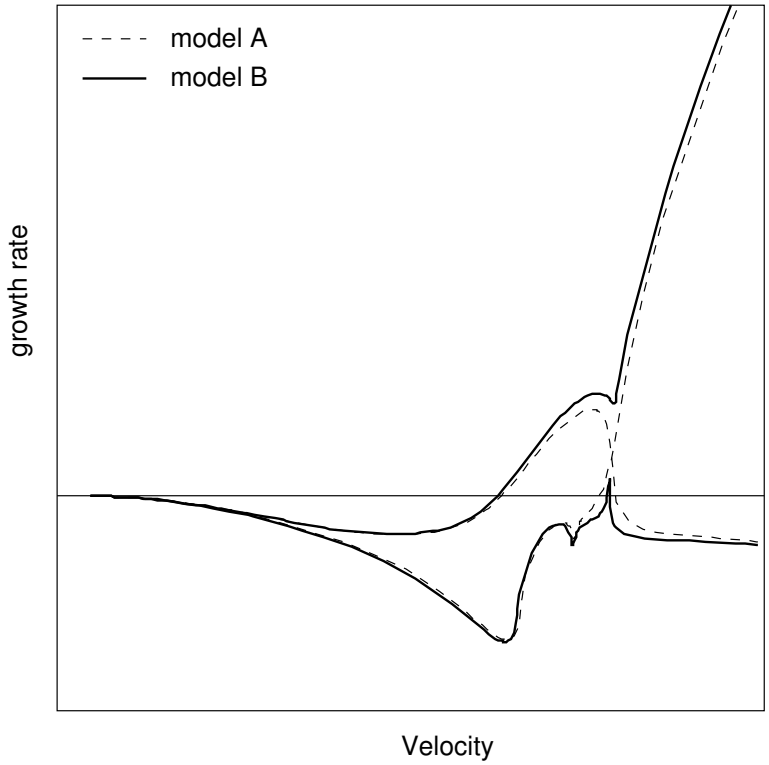


Figure 7.7: Suspected Mode Switching

for a constant-Mach solution, or

```
Finished tracking: min vtas (0) reached
```

for a constant-altitude solution, meaning the curve has reversed direction and ended up back at the start point. The sign of the determinant of the Jacobian matrix is used to safeguard against this, but it is not clear that this works in all cases.

Unexpected behavior in mode tracking is often mistaken for mode switching or reversal; on closer inspection it is almost always the case that the algorithm has performed as it should, keeping continuity in the curve. For example, a "slight" modification in an airplane flutter model caused the change shown in figure 7.7. The original model (model A) has two modes that flutter: a *hump mode* and a *hard* flutter crossing; in the modified model (model B) the two modes go unstable but in very different ways, and it appears that the algorithm has tracked the two modes incorrectly.

However, closer inspection shows that this is not the case. Figure 7.8 shows the suspect region zoomed in, where it is clear that the algorithm has kept continuity in the curve.

More revealing is a plot of growth rate against frequency in figure 7.9 where there is no

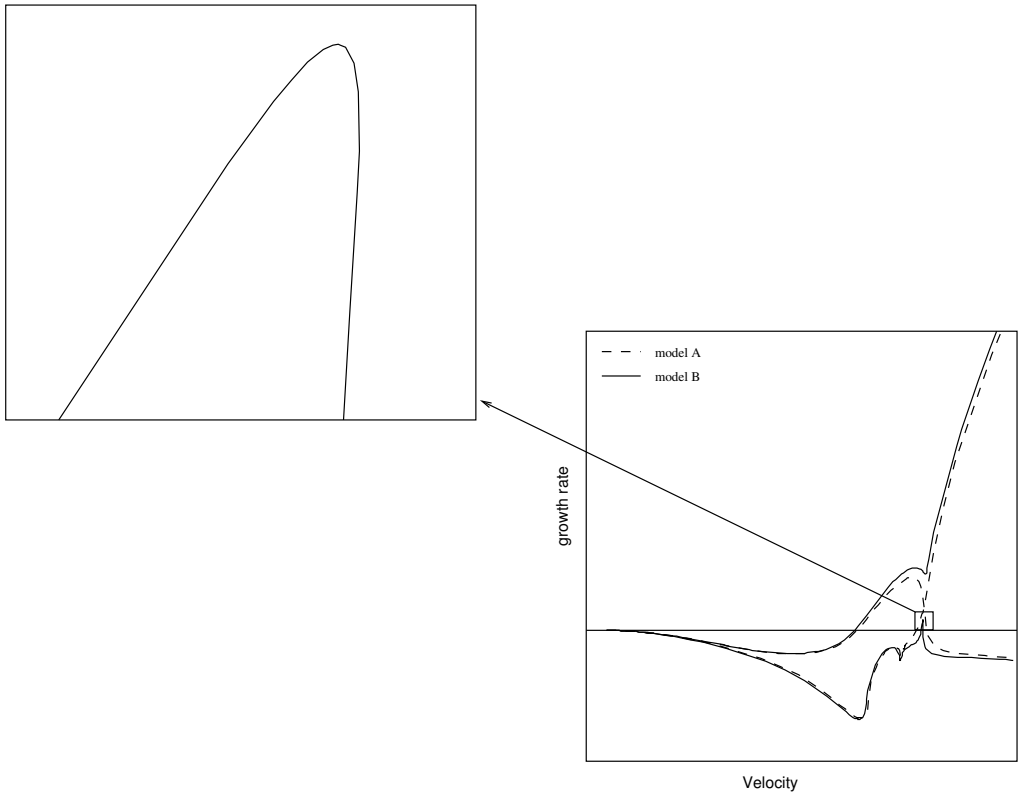


Figure 7.8: Suspect Region Expanded

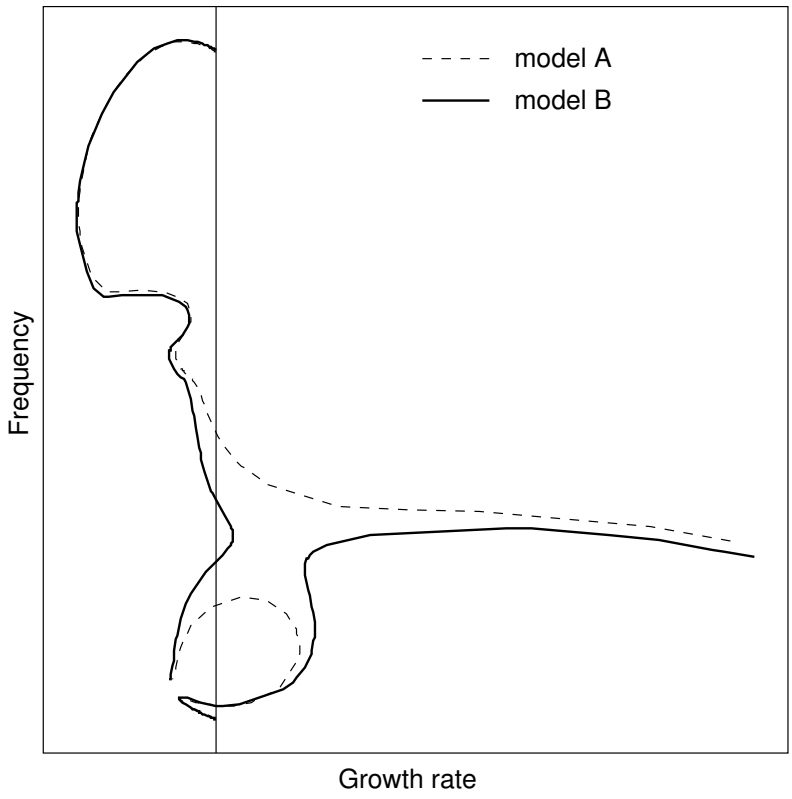


Figure 7.9: A More Revealing View

longer the spike in the modified model, the curves are smooth and well-behaved, with no question of mode switching.

This example illustrates how seemingly small changes in a model can make dramatic changes in the way aeroelastic modes evolve, even as the actual flutter speeds change very little.

From this example we can glean a few rules for determining if there is mode switching. First, zoom in on the suspected region. This is usually enough to determine if there is a problem, otherwise try plotting different combinations of parameters. For neutral-stability runs plotting sigma or growth against freq is often very instructive. Unfortunately neither task is as straightforward as it should be in pegasus; The FLAPS vis command 10.21 is much more conducive to exploring data, with the added advantage of using color to distinguish different curves.

7.8.1 Controls Equations

Other problems are peculiar to models with added controls equations and the nonlinearities they introduce. We mention two such problems from the FLAPS demonstration problems (§9).

The first problem is demonstrated by stab12.ax which has a control law represented by a user-written subroutine. Like most controls equations in user subroutine this one contains several rational (ratios of two) polynomials in the characteristic exponent s. The roots of the denominator polynomial are *poles* of the control equation, where the term goes to infinity. If the value of s. approaches one of these roots numerical problems will likely cause some sort of tracking problem, as it does in stab12.ax.

Another problem, extremely rare but potentially very serious is illustrated by demo problem stab13.ax: a mode that does not appear under 100 knots thus does not appear in an ordinary neutral-stability analysis. This example includes a control law, making the flutter equation more nonlinear with respect to velocity and characteristic exponent. It seems highly unlikely that this behavior could occur without the nonlinearities introduced by the controls equations.

Chapter 8

Visualization

8.1 Introduction

Visualization capabilities include graphical presentation of matrices, 2D plots of flutter and response results, matrix parameterizations, control-system eigenvalues, and 3D animations of flutter, response and vibration modes.

8.2 Visualizing Matrices

Large (sometimes even not so large) matrices can be viewed very effectively using the FLAPS print command or the matview option to the apex command. These both use a graphical program called MatView [22] developed at Oak Ridge National Labs. Figure 8.1 shows a typical MatView display.

Left-clicking on a matrix element displays the value of the element; a middle-click and drag zooms in the swept area, and a right-click unzooms. There are many more options; two of the more useful ones are a slider to change the color map and an option to view absolute values.

8.3 2D Plots

Two dimensional plotting is the most common type of plotting with dynamic analyses for good reason: visualizing more than two dimensions on two dimensional media like a computer screen or paper is very difficult, even with good shaded, textured, colored graphics. 2D plotting software is therefore very important. In FLAPS you have two choices: the FLAPS command vis or the Boeing program pegasus [34]. pegasus is good for creating documentation-quality plots with Boeing title blocks but it is lacking when it comes to exploring data with color, zooming, or viewing multi-dimensional data.

FLAPS processors stab, fresp, and tresp produce files which can be viewed using either

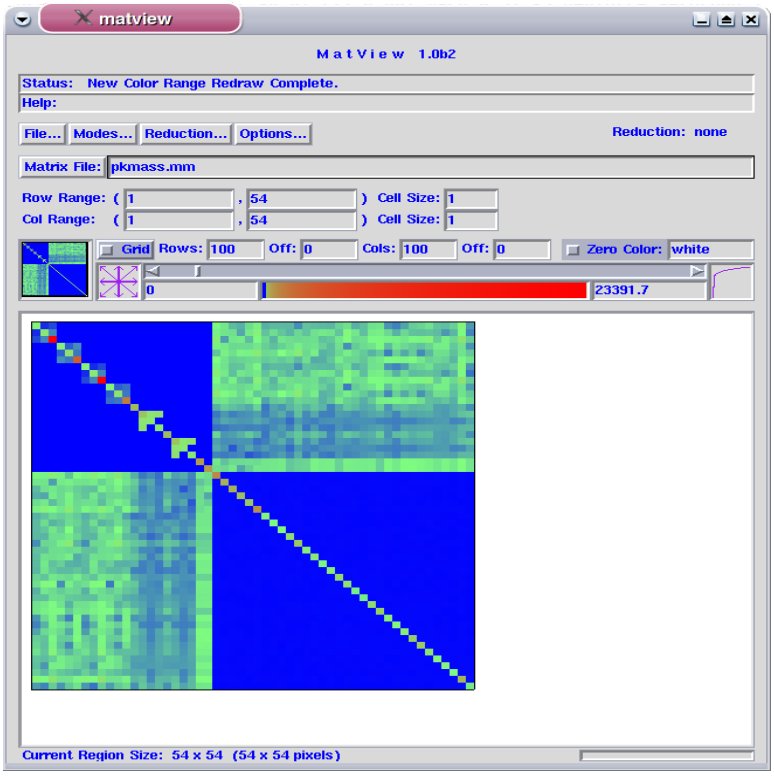


Figure 8.1: Matrix Image

PEGUSUS or vis; vis can be used to view results directly from an FLAPS database. vis can be used either in an FLAPS control program or on the command line; for example, to plot all matrix elements from the plot option in the param command:

```
$ apex vis gaf*.esa
```

or to view them one at a time:

```
$ for i in gaf*.esa;do $ apex vis gaf*.esa; done
```

A typical window from vis is shown in figure 8.2. The Hardcopy button allows for printing, creating a PostScript file, or creating a `fig` file which can be used in the drawing program xfig. Most figures in this manual were created using xfig to enhance plots created with vis.

Left-clicking on a vis window and dragging the mouse creates a new window with the swept area. This is how you zoom in vis. New windows created this way are independent; you can close any window without effecting the others or zoom in the new window.

Middle-clicking on a vis window brings up a display of the values of all other parameters at the point closest to the cursor. Figure 8.3 shows a typical display from a middle-click.

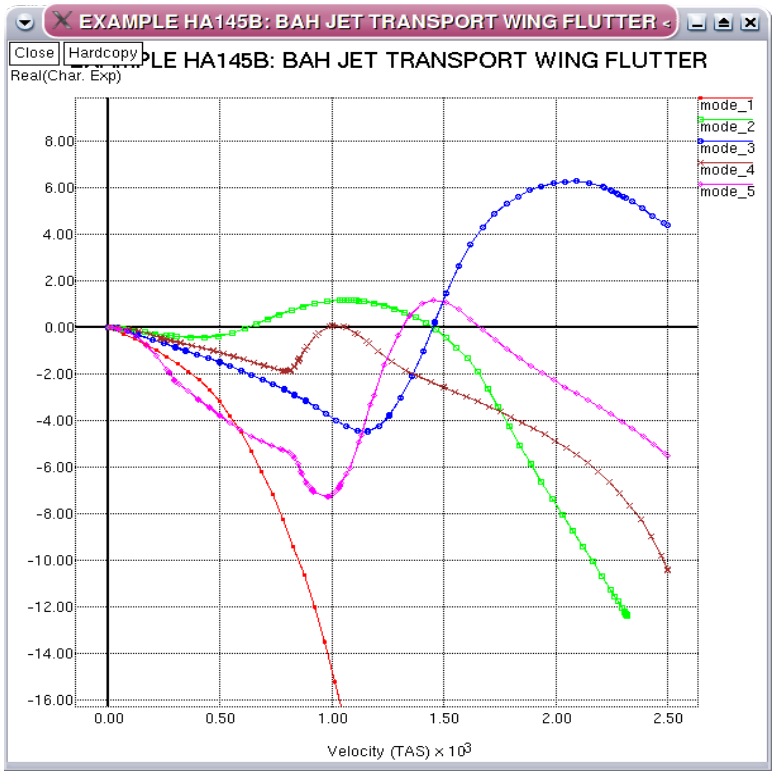


Figure 8.2: Typical neutral-stability plot

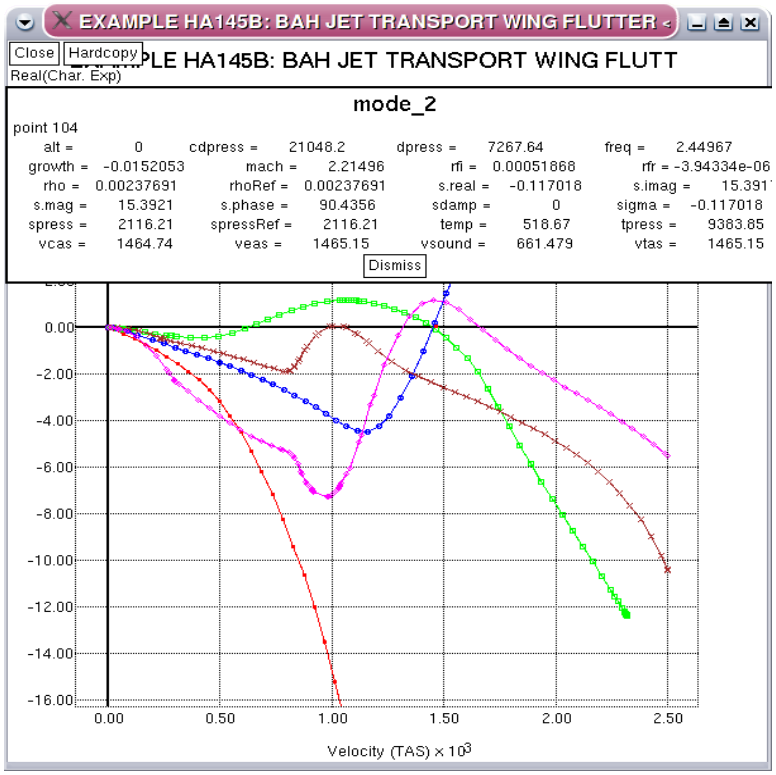


Figure 8.3: Parameter values from a middle-click

The right mouse button is used to pick points at which you want to view 3D animations of the motion of the structure. This feature is only available when vis is run from within an FLAPS control program, and if the necessary structural data is available: modes, nodal data, and (optionally) nodal connectivity data. Both vis and amvis are available as command line programs (§10.21.5).

8.4 3D Animated Modes

Motion of a structure can be visualized either by creating a file which can be used in amvis on the command line, or by right-clicking on a 2D vis plot of data from stab, fresp, or tresp at the point where the motion is to be visualized. The first method has the advantage of producing a file which can be used over and over and transferred to other platforms.

The second method is interactive in the sense that multiple points can be clicked on to visualize motion at multiple points. Thus the first method is useful when you know what conditions (e.g. `veas`, `growth`, or `freq`) you want to visualize beforehand, while the second method is useful when you are unsure of the conditions.

8.4.1 Creating A Universal File: The amv Command

If you know beforehand the conditions where you want to visualize the motion, or to visualize vibration modes the amv command (§**??**) can be used to either create a Universal file ([25]) or to start the FLAPS visualizer (amvis), or both. Creating plot files is often more convenient when running in a batch environment. Visualizing vibration modes requires nodal information in addition to the modes matrix; these are typically specified by a set number which tags all the necessary data.

In addition to the modal and nodal data, visualizing motion from a flutter or response solution requires the id of the solution and the values of enough parameters to uniquely specify the flight condition(s). For example, the following command could be used to create a Universal file of animated modes at flutter crossings from stab analysis `pk`:

```
amv { set=1, id=pk, o=pk.uf, growth=0 }
```

The resulting Universal file can be viewed with the FLAPS amvis command (§10.21.5), ufViewer, or X-Modal.

As another example the command

```
amv { set=1, id=pk, o=coupled.uf, veas=0 }
```

is one way to visualize so-called "coupled" modes (§3.8).

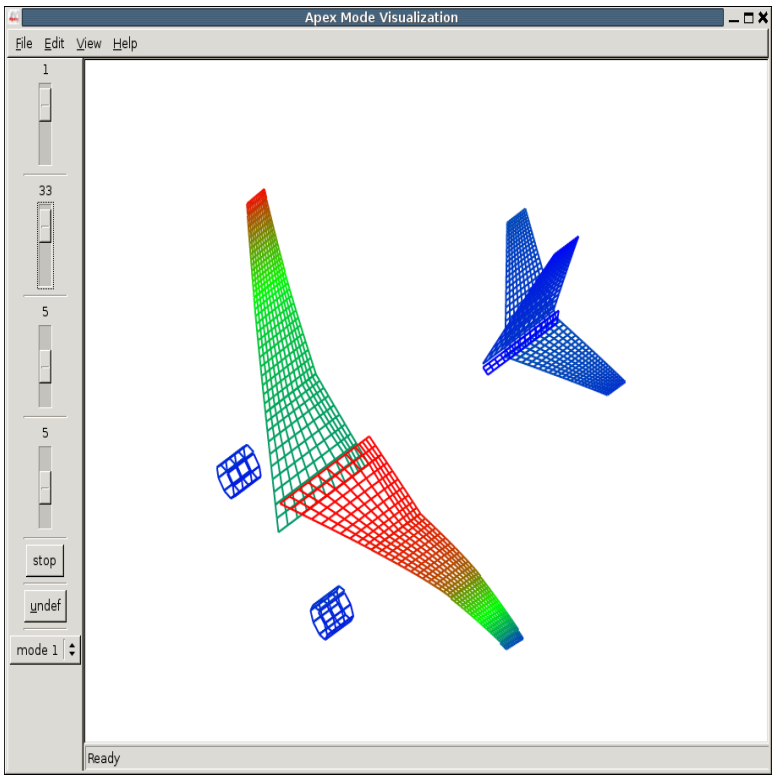


Figure 8.4: amvis with an aero grid

8.4.2 Visualizing Interactively

The parameter values where you want to visualize motion can be quickly and easily chosen by right-clicking on a 2D plot from vis. Following a right mouse click, the closest solution point is found and the corresponding generalized-coordinates are used to compute the motion of the structure as a function of time, and fed to the FLAPS 3D animated visualization program, amvis. A screenshot of amvis is shown in figure 8.4. In this way the motion of the structure at many different flight conditions may be viewed very quickly. The important point about this capability is that it is **interactive** in the sense that you can view motion at a flight condition, then based on what you learn from that visualization, choose another flight condition to look at and get immediate response. No files to transfer or programs to start up.

8.4.3 Visualization Transformation

Stability and response solutions are usually solved for in terms of generalized coordinates (§3.2) which must be transformed to physical displacements before the motion can be visualized. In addition, solutions done in the frequency domain (stab or fresp) must be transformed to the time domain. This sections explains how this is done in amvis or amv.

The stab and fresp processors compute, as part of every solution, a set of complex

generalized-coordinate amplitudes $\boldsymbol{q} = \boldsymbol{q}^r + i\boldsymbol{q}^i$ The actual motion of the nodes of the structure as a function of time (eqn. 3.15) is

$$\boldsymbol{x}(t) = e^{\sigma t}\boldsymbol{\Phi}\left(\boldsymbol{q}^r \cos\omega t - \boldsymbol{q}^i \sin\omega t\right) \tag{8.1}$$

where $\boldsymbol{\Phi}$ is the set of modes used to reduce the number of degrees-of-freedom of the structure. These modes are usually either free-vibration modes of the structure, branch modes of the components of the structure, or a combination of component modes and assumed modes.

The physical displacements are related to the nodes and degrees-of-freedom of the structure by two arrays of data: a set of nodes containing the location and node numbers of each node, and an array of freedoms, one for each element of the physical displacement vector with the associated node number. From this information the location of each node can be computed at every instant in time.

An important aid to visualizing the motion is a set of lines connecting the nodes, usually contained in an array of node numbers called a *connectivity* matrix. If this array is available the lines are added to the picture connecting the displaced nodes. Unfortunately there is currently no easy way to create the connectivity matrix, except when the modes and nodes are for an aerodynamic grid.

Unsteady aerodynamic programs such as the doublet-lattice program used in BAP usually compute aerodynamic forces for a set of mode shapes, the same modes used to reduce the model to generalized coordinates. Doublet-lattice requires the motion at the quarter-chord and three-quarter chord of each box in a regular rectangular grid, the aero grid. The motion at the structural nodes is transformed to motion on the aero grid through a variety of interpolation and extrapolation methods which are beyond this discussion. It can be very useful to visualize the modes transformed to the aero grid for two reasons: to check the validity of the interpolation scheme used, and because the aero grid has a natural set of connectivities. Demo problem stab11.ax uses this grid for visualization. exim2.ax will demonstrate the same for a BAP model when extraction of the aero grid becomes possible.

8.5 The Future

Visualization, both 2D and 3D, is vital to the types of analyses we do. Effort put into improving our visualization capabilities has the potential of large improvements in productivity. Good visualization software should **encourage** you to explore the data; it should be easy and quick enough to use that it fits well with existing processes.

Unfortunately we have not had the resources to make improvements in years. pegasus is nearly twenty years old. It was written before X Windows became a standard for workstations and it has never been upgraded to utilize windows in any meaningful way. vis is an improvement in a number of areas but it has always been a sideline. X-Modal is more than 15 years old and could quit working with the next AIX upgrade. The replacement FLAPS command amvis will hopefully fill this gap. Along with the rest of FLAPS, the future of visualization is uncertain; write your congressman.

Part II

Reference

Chapter 9

Demonstration Problems

9.1 Summary

This chapter describes a number of sample problems designed to illustrate the use of FLAPS commands. All of the problems are available in the demo directory; the easiest way to access this directory is with the demo option to the apex command (§10.6). The FLAPS input files all have names ending in .ax, FLAPS savefiles have the extension .sf, NASTRAN OUTPUT4 files have the extension .op4, Elfini neutral files have the extension .nf, and ATLAS jobs for creating FLAPS savefiles have the extension .q.

You are encouraged to not only look at these sample input files but to copy them, run them, and modify them to suit your needs.

Besides providing examples of FLAPS commands, these files are run periodically to catch bugs before they show up in your jobs. Most of them contain statements, usually at the end of the control program meant to catch errors in the results. Many of the files check that targets in stab, fresp, or tresp are met; to this end there are statements like

```
bool ok = true;
vector<Real> values;
values.push_back(24.238121);
values.push_back(187.99167);
ok = checkTarget("pk", "growth", "dpress", values, 2) && ok;
if (!ok)
   exit(1);
```

These statement probably are unfamiliar; they are C++ statements which in a nutshell create a vector of expected values (of dynamic pressure in this case) then call a built-in function (`checkTarget`) to read the results from stab, fresp, or tresp (with `id=pk` in this case) and check the targets against the expected values. If they are met the function returns true, if not it returns false and the program exits with an exit code of 1 to signal failure.

The currently available FLAPS example input files are

alge1.ax Rayleigh quotient demonstrating matrix transposition, inverse, and

operations on diagonals.

alge2.ax Demonstrates matrix pseudo-inverse (inverse of a rectangular matrix), scaling matrices, builtin constants.

alge3.ax Using a user-written subroutine parameterized matrix in alge. Interpolation of a set of matrices as a function of two parameters. Inversion of a parameterized (cubic-spline interpolated) matrix.

alge4.ax Matrix algebra on parameterized matrices. Does an eigensolution and modal reduction of a set of generalized aerodynamic matrices.

Requires stab123.sf.

exim1.ax Creates a simple beam model in ATLAS, imports the data into FLAPS and does a basic flutter analysis.

- runs ATLAS from inside the FLAPS control program
- creates an FLAPS savefile containing mass, stiffness and aero matrices
- restore the model from the savefile
- p-k flutter solution
- checks targets: flutter crossings

exim2.ax Creates a simple beam model in BAP, imports the data into FLAPS and does a basic flutter analysis.

- runs BAP from inside the FLAPS control program
- imports mass, stiffness, aero matrices from the BAP databases
- imports model geometry from the BAP databases and runs the FLAPS dublat command to generate unsteady aero matrices for comparison with BAP dublat.
- p-k flutter solutions using BAP aero and again using FLAPS aero
- parameter variation using interpolated mass matrices
- checks targets: flutter crossings

exim3.ax Exports and imports data in Matlab, NASTRAN OUTPUT4, and Matrix Market formats.

Requires stab123.sf.

merge1.ax Extract rows from a modes matrix, then merge them.

Requires stab123.sf.

modes1.ax Adds local coordinate systems to a modes matrix and the associated nodes, and freedoms matrices.

Requires stab123.sf.

modes2.ax Creates rigid-body modes and visualizes them.

Requires stab123.sf.

fresp1.ax Frequency response with aerodynamics near a flutter point using a constrained-displacement excitation and a force vector.

Requires stab123.sf.

fresp2.ax Constrained-displacement frequency sweep with ABCD control-law.

Requires stab123.sf.

fresp3.ax Demonstrates multiple Constrained-displacement frequency-response.

Requires stab123.sf.

fresp4.ax 787 p-beta ABCD frequency response.

Requires stab7.sf.

stab1.ax Basic flutter solution on 5-dof cantilevered beam wing.

Requires stab123.sf.

stab2.ax Flutter solution on 5-dof cantilevered beam wing with ABCD control-law

Requires stab123.sf.

stab3.ax Flutter with various aerodynamic parameterization schemes: k-interpolation, rational-function approximation, and p-M interpolation.

Requires stab123.sf.

stab4.ax User-written subroutine controls-equation on 5-dof cantilevered beam.

Requires stab123.sf.

stab5.ax Flutter solution using Elfini neutral file, and NASTRAN OUTPUT4 file.

Requires stab5.nf and stab5.op4.

stab6.ax Flutter solution including gyroscopic matrix and gyro spin-rate parameter variation.

Requires stab123.sf

stab7.ax ABCD control-law with internal time delays on 787.

Requires stab7.sf

stab8.ax 747 flutter, parameter variations, and optimization.

Requires stab8.sf, built using ATLAS job stab8.q.

stab9.ax Demonstrates the use of user-defined parameter equations, including Mach number and true airspeed computed using alternative equations.

Requires stab123.sf

stab10.ax NASTRAN example HA145b from the MSC/NASTRAN Test Problem Library.

Requires stab10.op4

stab11.ax ATLAS beam model of the 757.

- Uses FLAPS dublat to create p-value unsteady aero.
- Demonstrates a constant-mach V-g type solution.
- Nacelle frequency variation
- aeroelastic mode visualization on the aero grid

stab12.ax User-written control-law subroutine on an Elfini model of the 777. Two modes in this example run into control-law singularities: places where the denominator of a control-law term goes to zero.

Requires stab12.sf.

stab13.ax User-written control-law subroutine on 747 which demonstrates some very unusual and dangerous behavior possible with control laws.

Requires stab13.sf.

stab14.ax Modal truncation convergence study starting with a 220 dof BAP model and comparing flutter speeds at various smaller reductions.

Requires stab13.sf.

tresp1.ax Time-domain response with RFA aero.

Requires stab123.sf.

tresp2.ax time integration of a forced damped pendulum showing chaotic motion [26].

tresp3.ax Time-domain response using Fourier transforms

Requires stab123.sf.

9.2 Details

Here we give details on some of the more complicated or unusual problems where the results are often as interesting or educational as the setup.

9.2.1 Stab13

This 747 model has a mass matrix that is parameterized on fuel loading by interpolation (§6.1) and a control law represented by a user-written subroutine (§6.4). Figure 9.1 shows a cmvd (table **??**) flutter solution at 80 and 84 percent fuel.

There is something very wrong about this plot - there appears to be one curve on the left but three on the right! The explanation of this plot is a cautionary tale of what

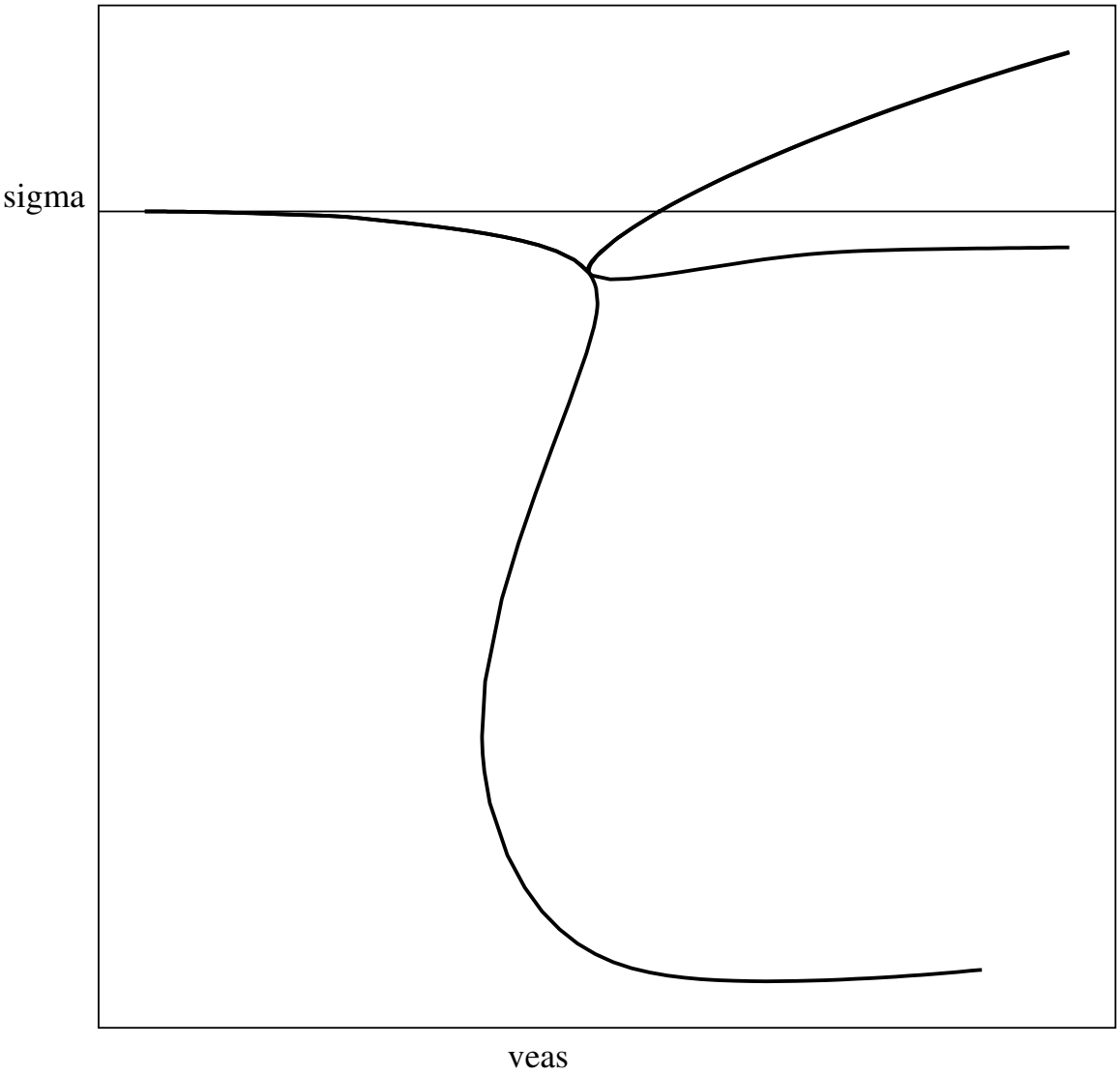


Figure 9.1: Strange Flutter Solution Curves

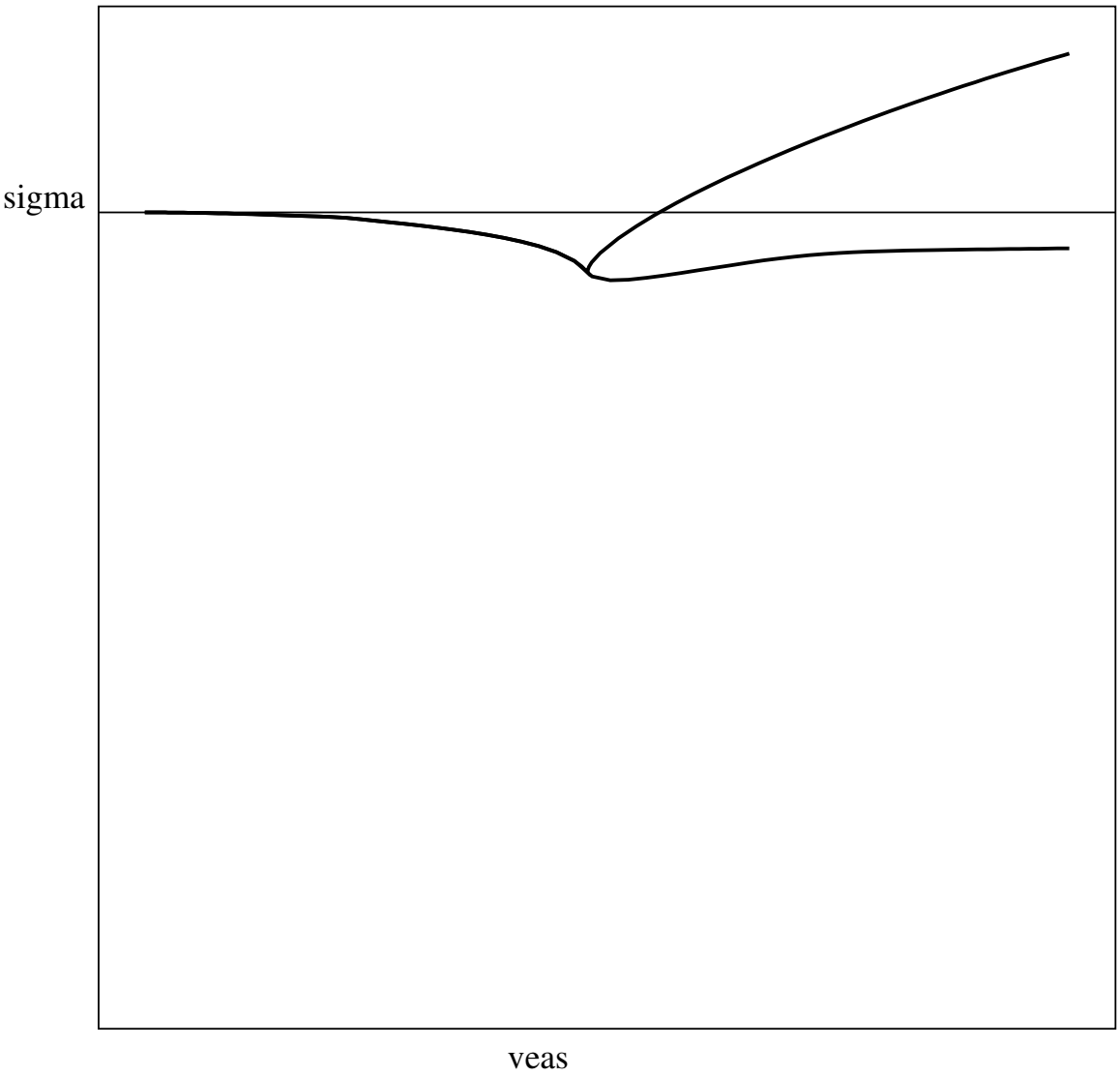


Figure 9.2: p-k at fuel = 80 and 84 percent

strange behavior can occur particularly with closed-loop systems, and what can be done about it. It begins with a typical sequence of analyses, a p-k flutter solution followed by a parameter (fuel) variation, only here we consider only one mode to simplify.

Figure 9.2 shows the mode at fuel loadings of 80 and 84 percent; at 84 percent the mode goes unstable but at 80 percent it is stable, not an unusual situation.

However when we do a fuel variation starting from the flutter crossing we see that this instability is insensitive to fuel (figure 9.3) so we would expect the mode at 80 to go unstable. Not shown are a number of other modes at 80, none of which are unstable, so the mystery is, what mode goes unstable at 80, since the parameter variation shows that there is one. To find it we take advantage of the fact that the FLAPS `stab` command can start tracking a mode from any previous solution, so we can set the active variables to `(veas, freq, sigma)`, start at 80 on the fuel variation curve and track back to zero airspeed.

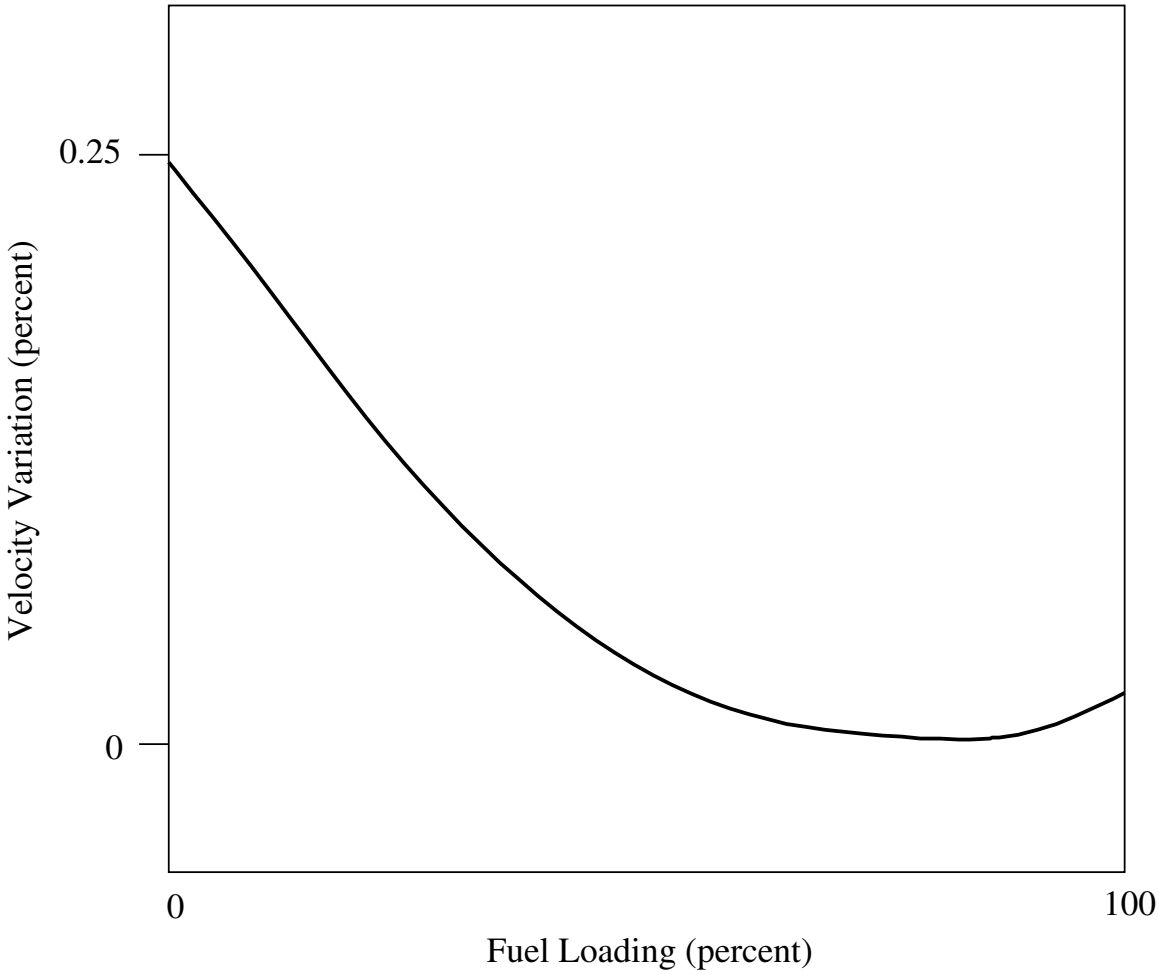


Figure 9.3: Variation of Flutter Speed with Fuel

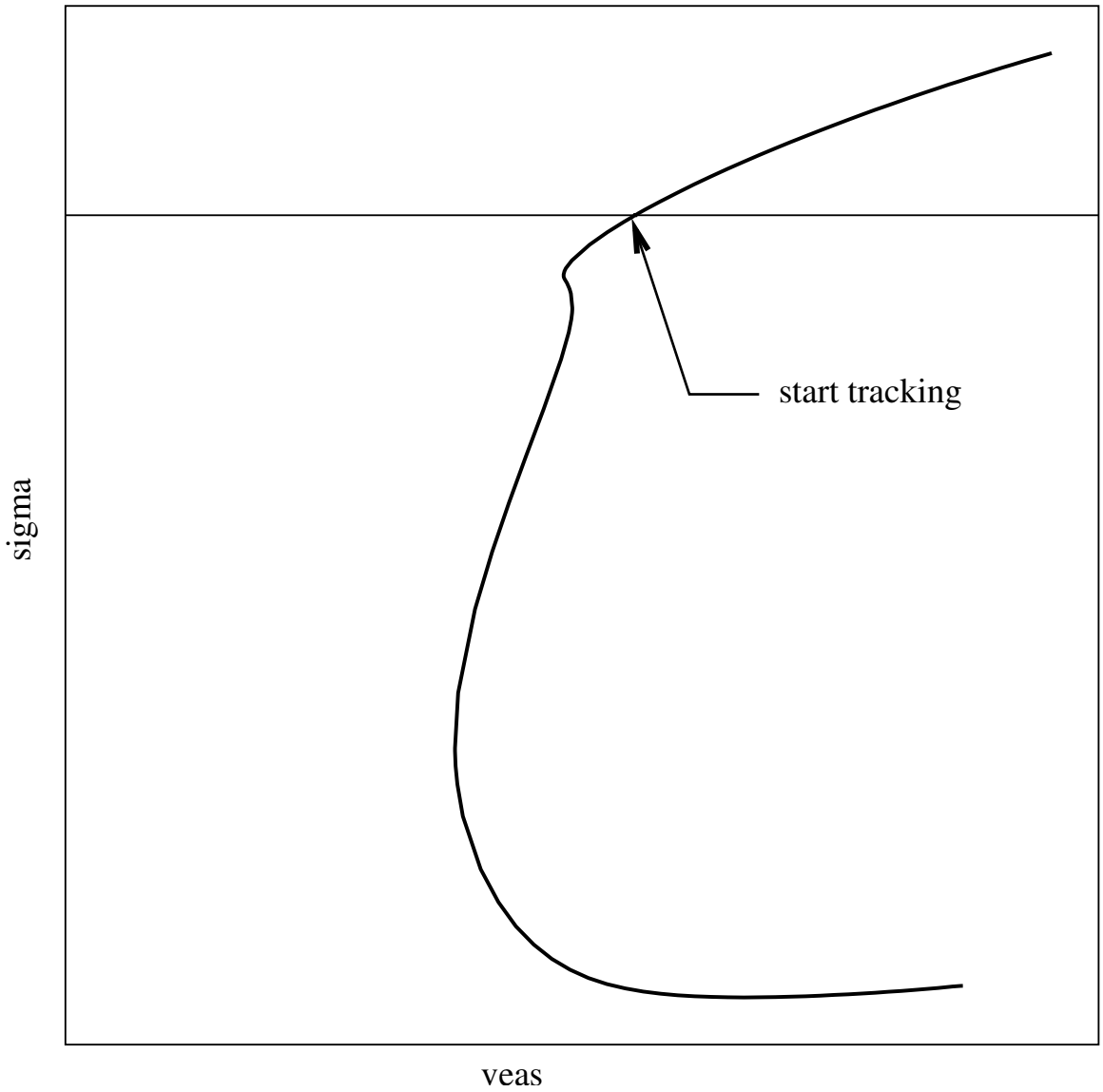


Figure 9.4: Tracking the Missing Mode

This is where it gets interesting because when we do this the resulting curve, shown in figure 9.4 does not go to zero, but instead turns toward increasing velocity. Had we done a flutter solution only at 80 we never would have found the instability because flutter solutions normally start at or near zero velocity. It was fortunately not a problem in this case because the engineer doing the analysis noticed the missing mode but it points out the need for guarantees on stability within specified flight conditions. In this regard topological degree theory ([35]) holds promise.

9.2.2 Stab14

`stab14.ax` is a modal truncation study: what happens when you reduce the size of the model using free-vibration modes? Starting with a BAP model with 220 component modes (§3.2.1) a simple flutter solution is performed tracking only one aeroelastic mode which goes unstable at a velocity we call V_0. Then the model size is reduced by computing eigenvectors of the free-vibration problem using the mass matrix at a different fuel condition (OEW). A triple-product reduces all matrices to the smaller size and the flutter solution is repeated with 180, 140, 120, and 80 degrees of freedom. The results are shown in figures 9.5 and 9.6. Two questions which might be asked about this study are 1) how many degrees of freedom need to be retained to get answers that are reasonably close to the original, and 2) how much computing time does trunction save? Figure 9.1 is an attempt to answer these questions. As expected the time drops dramatically as the size decreases; flutter solutions are dominated by numerical processes whose time inreases with the cube of problem size so an upper limit on the exponent is 3 and values less than 3 show how much of the solution process is consumed by bookkeeping.

dof	**size ratio** η	**time ratio** τ	**highest frequency Hz**	**flutter speed change**	**exponent** $\frac{\log \tau}{\log \eta}$
220	1	1	1406	1	
180	0.82	0.64	39.7	1	2.2
160	0.73	0.47	32.2	+0.007%	2.4
140	0.64	0.33	26.5	+2.5%	2.5
120	0.54	0.21	21.1	-2.8%	2.5
80	0.36	0.07	15.5	+8.0%	2.6

Table 9.1: Modal Truncation Results

Modal truncation for this model below 160 dof is probably unacceptable at 140 dof the flutter speed not only increases by 2.5 percent but the character of the mode changes dramatically, going from a “crasher” to a “hump mode”. Looking at the highest frequencies it is easy to see why modal truncation below 160 dof is unacceptable. At 140 dof the highest frequency has dropped to only 26 Hz, or only 2 to 3 times the flutter frequency. Evidently there are modes above 26 Hz which are important to flutter.

Another surprising aspect of this model is the number of free-vibration modes there are below 40 Hz. For comparison, stab8.ax is a beam model of a 747 which has only 52 modes below 20 Hz, while this finite-element model has 113! It would be very interesting to look at how many modes in this model are irrelevant to flutter, and if

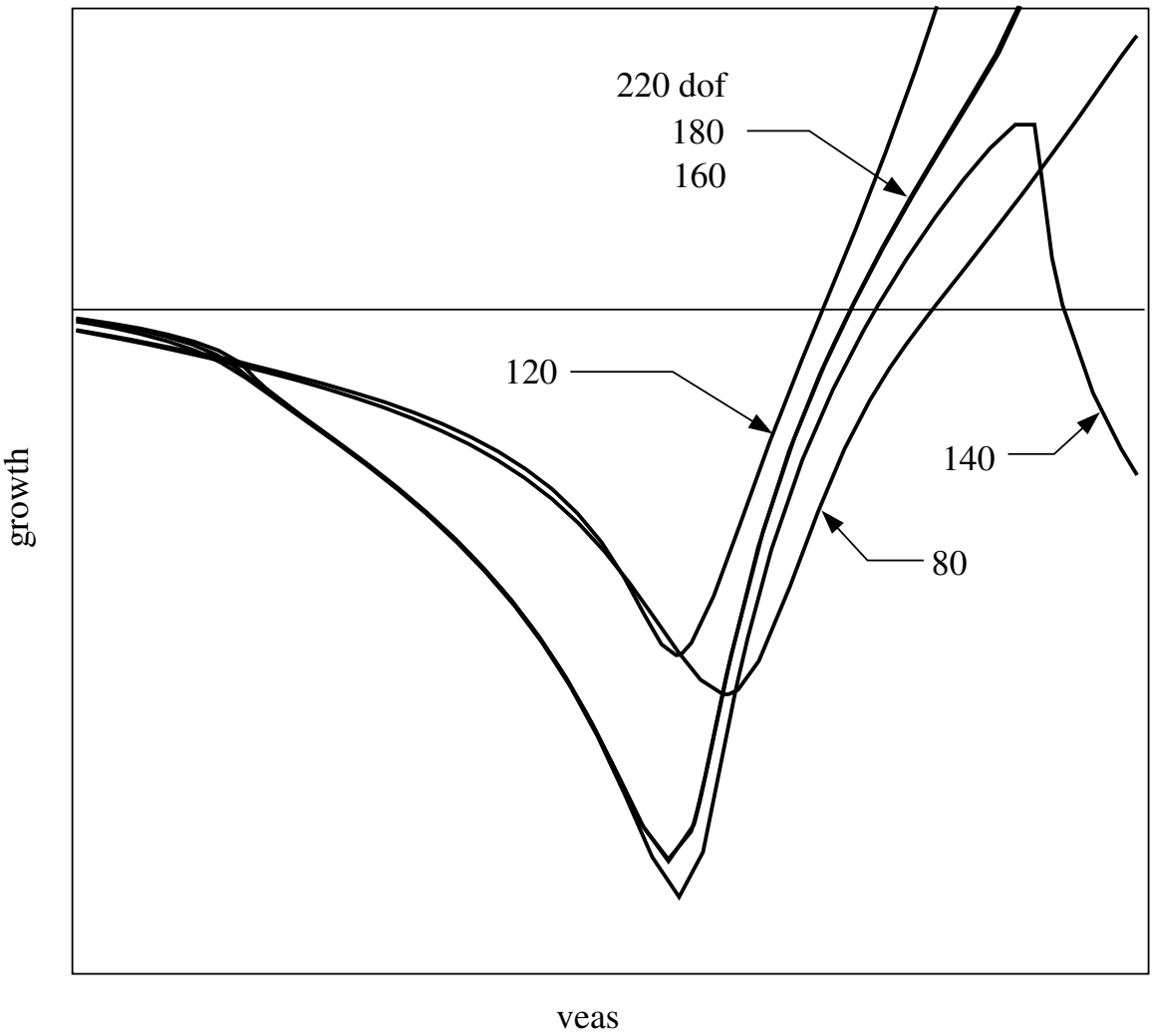


Figure 9.5: Modal Truncation: v-g view

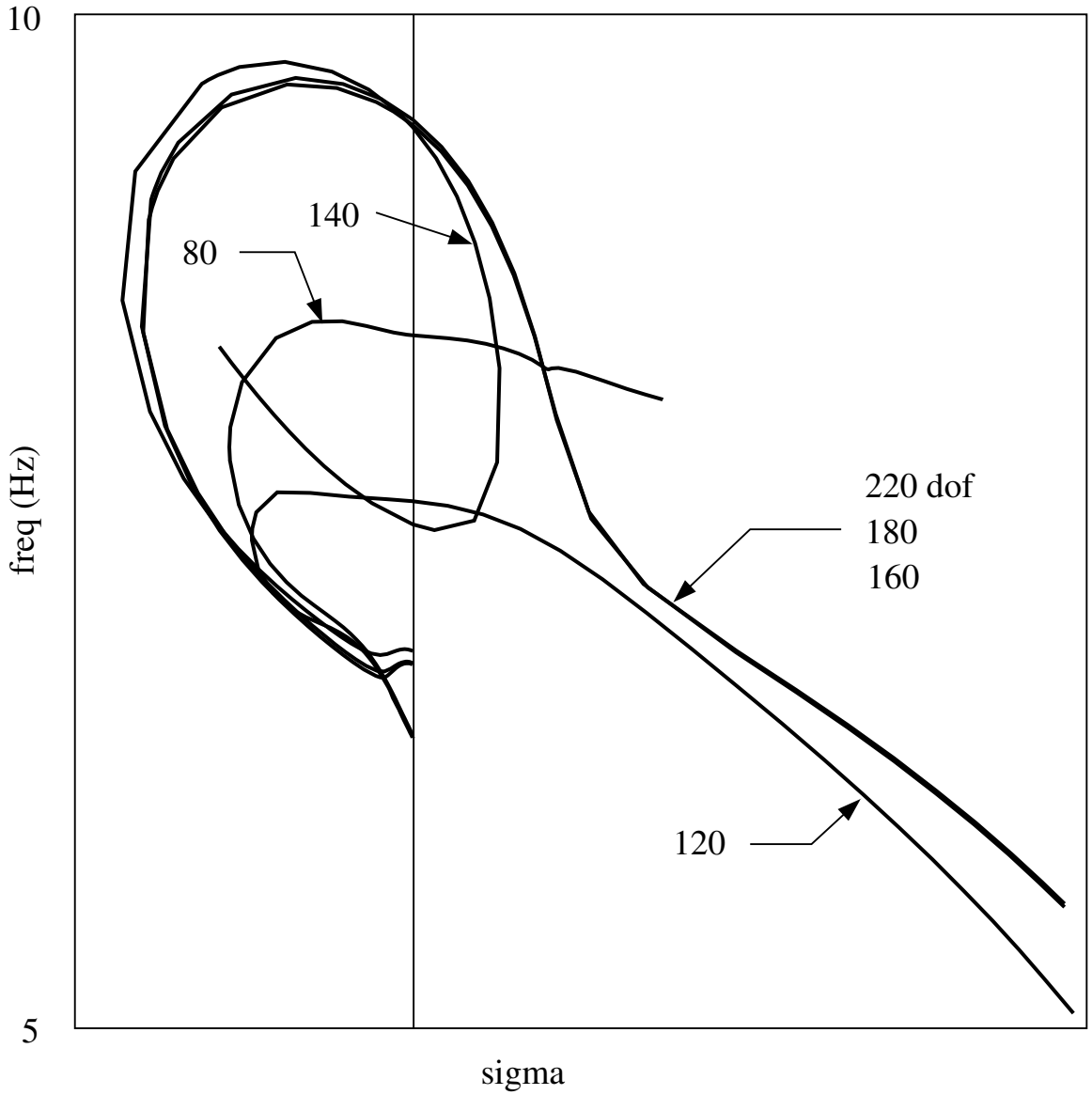


Figure 9.6: Modal Truncation: s-plane view

beam models are capable of predicting flutter as accurately as finite-element models at a fraction of the cost.

Chapter 10

Commands Reference

This chapter contains descriptions of all FLAPS commands. Descriptions are intentionally brief; theory is in part 1 and the appendices.

10.1 Syntax

One of the design goals for FLAPS has been to allow the user to write input files that are concise, easy to understand, and pleasing to look at. Rather than requiring strict formatting, FLAPS allows a great deal of flexibility in how input files are written, allowing the user to write in his or her favorite style. It is important to realize that writing FLAPS input files is "computer programming" and style does matter, for your own sake and for the sake of others who may have to read your computer programs later.

In general, the syntax of an FLAPS command has the form

command { *options* }

where command is one of the commands listed in this chapter. All FLAPS commands are case-sensitive: there is no Alge, only alge.

options are a set of comma-separated options of the form

keyword = value

Features of the option syntax include:

- keywords are case-sensitive: Mass=AEMHH is different from mass=AEMHH.
- if a keyword contains a comma it must be enclosed in (single or double) quotes to prevent the comma from being taken as an option-separator. The same is true of values.
- Some options do not take a value

- Whitespace (space, tab, or newline) characters are ignored except when enclosed in (double or single) quotes
- the comma separating options may be left off if an option is the last one on a line.
- in many cases the value (and sometimes even the keyword) may be a list with one of the following forms:

  (i,j,..k) Explicit list

  (i to j) Implicit list

  (i to j by k) Implicit list with a stride

  An implicit list with a stride is also known as a *to-by* list.

The value of an option may have one of the following datatypes:

string A character string of arbitrary length. If the string contains commas it must be enclosed in single or double quotes so that the commas are not taken as option-separators. As with the Unix shell, environment variables are interpreted within double quotes, but not within single quotes (see Examples below).

integer An integer

float A floating-point number, e.g. .7e+1, 7, 7.0 represent the same number

Each of these data types may be used in a list if the option allows; for example, here are pairs of equivalent options:

- *intlist*

```
eset = (77 to 89 by 3)
eset = (77, 80, 83, 86, 89)
```

- *floatlist*

```
freq = (1.0 to 2.5 by 0.5)
freq = (1.0, 1.5, 2.0, 2.5)
```

- *stringList*

```
plot = (freq, x1 to x10 by 2)
plot = (freq, x1, x3, x5, x7, x9)

print { ("AEMHH,MASSID=AAA" to "AEMHH,MASSID=AEA" by 2) }
print { ("AEMHH,MASSID=AAA", "AEMHH,MASSID=ACA", "AEMHH,MASSID=AEA") }
```

10.2 Keyword and Value Options: Option-Options

In some cases the keyword or value may be followed by options enclosed in curly braces. That is, options may themselves sometimes have options. For example

```
AEKHH{col=(33,34 to 40)}
node=9934{ orient=(0,0,1) }
hingegap=0.1{ nominal=6.28 }
```

Sometimes these options are one or more parameter definitions (§5.1), which may be as simple as setting a parameter value. For example

```
sdgc=(14,16 to 20){fwdfus=0.02}
```

or with a more complete description of the parameter fwdfus:

```
sdgc=(14,16 to 20){fwdfus(Forward Fuselage)[0:10]<HZPRS>=0.02}
```

10.3 Examples

Here are a few examples of options; more can be found in the demonstration problems.

Note the use of quotes to protect the commas in the matrix name:

```
alge{ "mass,fuel=0" = "AEMHH,MASSCOND=111"
      "mass,fuel=10" = "AEMHH,MASSCOND=112"
      "mass,fuel=20" = "AEMHH,MASSCOND=113"
}
```

Options following a value:

```
param { i=gstif001
   gc = 2{freq2(Second Mode Freq)[0:10]}
   sdgc = 2{sdamp2(Second Mode damping)[0:0.2] = 0.01}
}

extract { irulty = "modes,set=11"{row=621010/ty} }
```

Lists of values:

```
stab { id=pk
   active = (freq, vtas, growth)
   fuel = (0, 10, 20)
   rset = (1 to 50, 61, 62)
}
```

Environment variables are expanded when used in options; for example

```
output{environment="massmatrix=fuelcnt"}
stab { ..., mass = $massmatrix, ... }
```

If it is necessary to enclose an environment variable in quotes, use double quotes if the variable is to be expanded:

```
output{environment="fuel=9010"}
stab { ..., mass = "AEMHH,MASSID=$fuel", ... }
```

is equivalent to

```
stab { ..., mass = "AEMHH,MASSID=9010", ... }
```

but with single quotes

```
stab { ..., mass = 'AEMHH,MASSID=$fuel', ... }
```

fuel is not replaced with 9010 because environment variables are not replaced within single quotes, just like the Unix shell.

10.4 Printed Output

Printed output presents a dilemma in most structural-dynamics programs: too much printout makes it difficult to find the important information; too little printout risks missing the important points. Warning messages are often missed because of too much printout.

Two techniques used in FLAPS to reduce the amount of printout without sacrificing important details are graphic display of matrices, and a one-line summary of generalized-coordinates and other vectors.

10.4.1 Matrix Display

Matrices are notoriously difficult to present in printed output. The approach taken in FLAPS is to present all but the smallest matrices graphically. The print command (§10.16) will display a matrix in a separate window with colors indicating the size of individual elements. Other FLAPS programs write matrices to files with .mm extensions ([24]) which can be displayed by the matview option to the apex command (§10.6). Figure 10.1 is an example of a matrix displayed in matview.

10.4.2 Generalized-Coordinate Summary

At various places in the printed output generalized-coordinates (or other vectors such as energy) are summarized in a single line by showing which element has the largest

magnitude, followed by the second largest, and so on. Next to each element number is a decimal fraction of the largest component in parentheses. A typical summary looks like this:

```
28  20(0.39)  19(0.32)  24(0.25)  29(0.24)  41(0.16)  68(0.16)  7(0.13)   47(0.13)   67(0.12)
```

which means that generalized coordinate 28 has the largest (absolute) value, and that value is 1.0. The next largest is 20 which has a magnitude of 0.39, followed by 19 with a magnitude of 0.32, and so on. Occasionally you may see a summary like

```
[0.82383]28  20(0.39)  19(0.32)  24(0.25)  29(0.24)  41(0.16)  68(0.16)  7(0.13)   47(0.13)   67(0.12)
```

The extra number in square brackets [0.82383] means that the largest value in the vector is not 1.0 but 0.82383. Note that the numbers in parentheses have not changed; this is because these numbers are actually the magnitude of that generalized coordinate relative to the largest, so for example the magnitude of generalized coordinate 20 is $0.82383 * 0.39 = 0.32129$.

10.5 alge

Performs various algebraic operations on matrices.

10.5.1 Syntax

alge { *options* }

10.5.2 Options

See §10.1 for the general syntax of options.

Options to alge consist of one or more equations, one per line. The left side of the equation is the name of the output matrix; the right side is an expression with one or more of the following elements; the expression must be enclosed in (double or single) quotes if it contains commas:

scalar expression Scalar expressions can be as simple as a number (e.g. 1.3e-4) or more complicated, using arithmetic operators (+-/*), square root (sqrt), powers (pow), or exponential (exp). Numbers may be real (e.g. 1.e-3) or complex. Complex scalars must be enclosed in parentheses and have an 'i' or 'j' before or after the imaginary part. For example (1.2+3i), (1e-4+j6.3) and (3+i3) are legal complex scalars.

Pre-defined scalars (§2.5) include

HZPRS	conversion factor from rad/sec to Hz ($1/\pi$)
KPIPS	conversion factor from inches/second to knots
DPR	conversion factor from radians to degrees ($180/\pi$)
RPD	conversion factor from degrees to radians ($\pi/180$)
G	acceleration of gravity in $in/sec = 386.0885826$
PI	$\pi = 3.14159265359....$

Scalar expressions may be used to create a named temporary scalar which may then be used in subsequent matrix expressions. For example, to define a variable `scale` and use it you could include

```
alge { scale = 1/sqrt(37.4)
   M = scale*AEMHH
}
```

matrix Name of an FLAPS matrix, or the name (and possibly some of the attributes) of a matrix or set of matrices with attributes (§2.3). Operations will be performed on all matrices with this name and attributes (see Examples). The name may be followed by options enclosed in curly braces:

row = *intList* Rows which are to be operated on; only used when a matrix is multiplied by a scalar, for example in the expression 3*GMASS{row=(3 to 10 by 2), col=(1,2)} all elements that are in rows 3, 5, or 7 **and** columns 1 or 2 will be scaled; that is, only elements (3,1), (3,2), (5,1), (5,2), (7,1), and (7,2).

Default: all rows

col = *intList* Columns which are to be operated on; see row option for restrictions.

Default: all columns

parameter = value Specifies values for some or all of the parameters the matrix is a function of. If the matrix is a function of parameters and no values are specified, operations on the matrix apply to the full range of all it's values. For example, if a is a function of 2 parameters, the equation b = a(-1) results in a matrix b which is a function of the same two parameters, whereas the equation c = a(-1){p1=1,p2=2} results in a matrix c which is constant.

operators Operators on matrices include the arithmetic operators +, -, *, transpose (t), inverse (-1), inverse-transpose (-t), and conjugate-transpose (*). Transpose, conjugate-transpose, inverse, and inverse-transpose follow the matrix name, e.g. GMASS(t). Scalars may be used to multiply or divide each term in a matrix. A scalar multiplying a matrix can pre- or post-multiply the matrix; a scalar divisor must appear after the matrix. See Examples.

identity The identity matrix may be included using upper-case I. The size of the identity is determined by the other matrices in the expression; e.g. in the expression K - I, the size of I will be the same as the size of K. Alternatively, the size may be specified by including the size option on I, for example I{ size=10 } results in a (10,10) identity matrix. The only time it is necessary (or recommended) to include the size option is if there are no other matrices in the expression from which the size can be deduced, for example in the expression A = 3*I{ size=5 } which produces a (5,5) diagonal matrix of threes.

diagonal The word diag has a special meaning when followed by a matrix name or expression in parentheses. If the matrix is a vector (i.e. an n by 1 matrix), a new diagonal matrix is created from the given matrix where the diagonals of the new matrix are the elements of the vector. For example

```
alge { a = diag (sqrt(2)*b) }
```

will create a diagonal matrix (a) with the elements of the square-root of 2 times b on the diagonal.

real The word real followed by a matrix name or expression in parentheses will create a new real matrix from the real part of a complex matrix. For example

```
alge { a = real (b+c) }
```

will create a real matrix (a) made from the real parts of the elements of the sum of b and c.

imag The word imag followed by a matrix name or expression in parentheses will create a new real matrix from the imaginary part of a complex matrix. For example

```
alge { a = imag(b(*)*b) }
```

will create a real matrix (a) made from the imaginary parts of the elements of b conjugate-transposed times b. Incidentally, a will be skew-symmetric.

conj The word conj followed by a matrix name or expression in parentheses will create a new matrix which is the conjugate of the existing matrix or expression. For example

```
alge { a = conj((b-c)(-1)) }
```

will subtract c from b, invert the result, then put the conjugate into a.

MAC The word MAC followed by two matrix names or expressions separated by a comma and contained in parentheses will create a new real matrix which is the Modal Assurance Criteria of those two matrices. [3] For example

```
alge { a = MAC(b{col=(1,3 to 10)}, c{col=(2,3 to 10)}) }
```

will create a real matrix (a) made from the Modal Assurance Criteria of selected columns of existing matrices b and c. This matrix can then be visualized using the print command:

```
print { a }
```

which will produce a window like figure 8.1.

10.5.3 Examples

Several demonstration problems use FLAPS savefile stab123.sf which contains 45 matrices with the name genforce and various values of attributes rfr, rfi, and mach. Specifying some of the attributes means that the operation will be applied only to a limited subset of the matrices; for example

```
alge { gf = 2*"genforce,rfr=0"{ col=(1 to 6 by 2) }
```

multiplies columns 1, 3, and 5 of all matrices with the name genforce and attribute rfr=0 by 2.

```
alge { a = DPR*((x(t)*b(-1)*x) + c) }
```

will invert existing matrix `b`, pre-multiply by the transpose of existing matrix `x`, post-multiply by `x`, add the result to `c`, convert it from radians to degrees, and write this result out as matrix `a`.

```
alge { t = "PI*sqrt(1/3) * pow(3, .5)"
       a = t*b
       c = t*t*b
}
```

will create a temporary variable `t` (which reduces to π), multiply `b` by `t` to produce `a`, and by `t` squared to produce `c`. Note the first expression is quoted to protect the comma.

```
alge { a = (G + c)/@G }
```

adds two matrices and divides the result by the acceleration of gravity; note the use of the @ character to prevent confusion with the matrix G.

Complex scalars can be used to create complex matrices out of real ones. For example

```
alge { modes = "(1+i0)*m1 + (0+i1)*m2" }
```

creates a complex matrix (modes) comprising m1 as the real part and m2 as the imaginary part.

10.6 apex

Reads an FLAPS control program consisting of one or more FLAPS commands and executes them. Commands may be in a file, from standard input, or may be entered interactively.

10.6.1 Syntax

apex [-h] [-v *version*] [-r*n*] [*filename* | *keyword*]

10.6.2 Options

See §10.1 for the general syntax of options.

If there are no options on the command line, apex enters *interactive mode* where any apex command may be typed and the command is executed as soon as it is entered. A line containing only the word end or quit finishes the interactive session.

-h

Prints a short summary of the options for the flaps command; for more help type apex help which displays the FLAPS User's Manual.

Default: none

filename

Name of a file containing the FLAPS input file. See chapter 2 for the structure of this file.

Default: if no *filename* or *keyword* is given the behavior depends on whether the flaps command is entered from the command line or within a script (e.g. a qsub file): if the flaps command is entered from the command line, apex enters interactive mode. If the flaps command is run from a script, input is from standard input, for example from a Unix *here* document (§2) (see example below).

Various utilities are available using the following *keywords* instead of a *filename*:

atmos

interactive program written by Boeing employees T.A.Monson, R.D.Johnson, and R.J. Tushoski which calculates properties of the standard atmosphere. Usage instructions are in the help menu accessed from the opening screen.

catalog *file*

list the matrices on FLAPS savefile *file*.

clean [-r]

remove any FLAPS temporary directories left from previous runs. Used with -r this command cleans this and all directories below it (recursive clean)

demo

Change the current working directory to the FLAPS demo directory where a number of example jobs are kept along with savefiles necessary to run them. Copy these to your own directory to run them yourself or modify them to suit your purposes.

diff *filea fileb*

create a Matrix Market (.mm) file which is the relative difference between the matrices contained in two .mm files.

dtree [-s] [*path*]

display on stdout the directory tree rooted at the current directory or *path* if included. The -s option prints the directory size next to the directory name and gives a summary of the largest directories.

dog

Works like the Unix cat command except that environment variables are replaced

extract *options*

Extract matrices from an Elfini database. Typing `apex extract` (no options) will list the usage instructions.

help

opens the FLAPS User's Manual in Adobe Acrobat Reader.

import

creates an FLAPS savefile from a BAP or NASTRAN database. (provides the same functionality as the obsolete program nastrap).

kill *job-number*

kills a running FLAPS job with *job-number* given by the apex spy command.

matview *file*

Visualize a matrix in a file with a `.mm` extension. These files are usually produced by the FLAPS print command when matrices are too large to include in stdout. A program called MatView is used to visualize the matrix. MatView has a number of features which make examining the structure and values of a matrix very easy, including color scaling, zooming, and viewing individual matrix elements.

rename *regular-expression replacement* [*file(s)*]

Renames multiple files based on a regular-expression (appendix I) (§I), a string to replace what the regular-expression matches, and an optional list of files to consider (the default is all files in the current directory). For example, if a directory contains

```
C8D3PLC2M070C000M75vG39_lgs070912_sd00_08.apx
C8D3PLC2M070C000M85vG39_lgs070912_sd00_08.apx
C8D3PLC2M070C000M95vG39_lgs070912_sd00_08.apx
```

you could change G39 to G40 with

```
apex rename G39 G40 *.apx
```

For an example requiring some knowledge of regular-expressions consider how to rename files containing the strings M75, M85, and M95 to M70, M80, and M90, respectively:

```
apex rename 'M\(.\)5' 'M\10' *.apx
```

Here the regular-expression is enclosed in single quotes, which is almost always necessary to protect it from the shell interpreting it. Note that the list of files must not be in quotes because we want the shell to expand it to all files ending in .apx. The replacement string contains a backslash followed by the number one, which means "replace with the first match in the regular-expression". Matches in the regular-expression are the parts enclosed in parentheses (escaped with backslashes), in this case just a period, which means "any character". So this rename literally means "replace strings that contain the letter M followed by any character followed by the number 5 with M followed by the same character as in the original, followed by the number 0".

spy

list the status of all processes for all of your currently running FLAPS jobs

ufv *filename*

visualize animated modes using ufViewer

vis [*options*]

Run a simple 2D plotter (the FLAPS vis command). See the vis reference (§10.21.5) for details on usage.

amvis

visualize animated modes using data contained in the specified Universal file ([25]).

The following options are used to specify alternative versions of FLAPS such as pre-release or obsolete versions:

-v *version*

FLAPS *version* to use (prod, beta, or alpha). The default, prod, is the current production release. beta is the upcoming release which is currently undergoing beta testing, and alpha is the current development version, hence the least stable version. See the discussion above for the difference between an FLAPS version and release

`-r`*n*

use FLAPS release *n*, for example -r3.7. Note the difference between an FLAPS release and an FLAPS version: version refers to a stage in the lifecycle of an FLAPS release, so for example release 3.7 began as the beta version, and during this period `-r3.7` and `-v beta` were identical. Later when 3.7 became the production release `-r3.7` had the same effect as the default `-v prod`.

10.6.3 Input and Output

Input is either a file containing FLAPS commands or a set of commands in the standard Unix input stream *stdin*. If FLAPS is run interactively by typing `apex` at the command prompt, stdin is from what you type; if FLAPS is run from within a script (perhaps in batch mode) stdin is conveniently taken from a Unix *here document*. A `here document` begins with the << operator followed by a string of characters of your choice; for example

```
   apex <<@
      import{myfile}
@
```

Here documents are convenient because the input to FLAPS can be placed in the script without creating a separate file, and environment variables are substituted, so a script like

```
export SAVEFILE=/home/me/myfile
   apex <<@
      import{$SAVEFILE/myfile}
@
```

is equivalent to

```
   apex <<@
      import{/home/me/myfile}
@
```

Output from command/apex/ is normally to the standard Unix output streams *stdout* and *stderr* unless redirected or the `output` command is used to redirect stdout and/or stderr. See any reference on the Unix operating system for more details, for example [16], [1], [29], or [44].

10.6.4 Examples

To run FLAPS from within a script, taking the input from the same script, use what is known as a Unix `here` document [44]:

```
apex <<EOF
    restore{savefile}
       ...
    end
EOF
```

10.7 catalog

Prints a catalog of FLAPS matrices matching specified conditions.

10.7.1 Syntax

catalog { *options* }

10.7.2 Options

See §10.1 for the general syntax of options.

By default, catalog prints a catalog of all FLAPS matrices; the list may be limited by specifying a list of matrix ids (see the explanation of matrix ids in section 2.3. or regular-expressions (see section I).

10.7.3 Output

output consists of a list of the matrices matching the matrix ids specified, printed on the standard output file.

10.7.4 Examples

To catalog all matrices:

```
catalog {}
```

To catalog all matrices with the mode=3 attribute:

```
catalog { mode=3 }
```

10.8 export

Writes FLAPS matrices to files that can be imported by other programs. Currently available formats are Matlab, Output4, and ESA. Universal files can be created with the FLAPS amv command, and Matrix Market formatted files can be created with the FLAPS print command.

10.8.1 Syntax

export { *options* }

10.8.2 Options

See §10.1 for the general syntax of options.

Options to export consist of the output file name, the format of the exported matrices, and either a solution id from a previous execution of stab, fresp, or tresp, or one or more matrix names to export.

o = *string*

Name of the file to receive the exported matrices. The filename extension determines the default export format:

.esa ESA format, ([34]) a format created by Boeing for use in the 2D plotting program Pegasus. The FLAPS command vis can also use ESA-formatted files. a format widely used in the numerical analysis community for distributing test matrices. Used for printing large matrices in FLAPS and viewing with matview (§10.6) [24].

.mat Matlab or Simulink [23] MAT-files are binary, hence not in general portable across platforms.

.op4 OUTPUT4 (NASTRAN or BAP) [37]

id = *string*

name of a previously generated solution from stab, fresp, or tresp.

parameter-defn

(ESA files only) the name of a parameter in the solution associated with the id option or a name to be given to data in a row of the specified matrix. This option may be repeated to specify all parameters that are to be included in the ESA file.

matrix-id

name of an FLAPS matrix to be exported; any number of matrix names may be included.

10.8.3 Examples

```
export { id = pk, vtas, freq, growth }
```

will create an ESA-formatted file named `pk.esa` of solution curves with the parameters `vtas`, `freq` and `growth`

```
export { o = stif.mat, AEMHH, AEQHH, AEKHH }
```

will create a file containing AEMHH, AEKHH, and all the unsteady aero matrices, which can be imported into matlab.

10.9 extract

Extracts rows, columns, or diagonals from an existing matrix, creating a new FLAPS matrix.

10.9.1 Syntax

extract { *options* }

10.9.2 Options

See §10.1 for the general syntax of options.

Options to extract consist of the input matrix name, the output matrix name, and lists of rows, columns, or diagonals to extract or if the matrix is square the rows and columns to retain or eliminate may be specified. Rows, columns, and diagonals may be specified either as integers or node/freedom: a node number followed by a slash followed by a freedom number in the range 1-6 or the corresponding text (tx, ty, tz, rx, ry, or rz). If nodes/freedoms are used it is necessary to include the set option unless set is an attribute in the matrix id.

Multiple extractions can be included, but to do so it is necessary to put those options which are specific to each extraction in curly braces following the input matrix name (see Examples).

In addition to the row, column, and diagonal element specification, curly braces following the input matrix name may contain parameter values. The input matrix will be evaluated at those values prior to extracting the matrix elements (see Examples).

output-name = *input-name* { *options* }

The output matrix name (left-hand-side) and the input matrix name (right-hand-side). The options which specify the elements to be extract follow the input matrix name and are enclosed in curly braces; this allows multiple extractions to be done in one extract statement. The extraction options are:

rows = *stringList*

a list of row numbers or node/freedom combinations. If a row number is zero, the output matrix will have a row of zeros.

col = *stringList*

a list of column numbers or node/freedom combinations. If a column number is zero, the output matrix will have a column of

zeros.

diag = *stringList*

a list of diagonal numbers or node/freedom combinations.

rset = *stringList*

a list of rows and columns to extract (square matrices only).

eset = *stringList*

a list of rows and columns to eliminate (square matrices only).

size = *int*

only relevant if the input matrix is the identity, this options specifies the order of the identity.

set = *string*

Nodal data set identifier; only necessary if the rows and/or columns are specified as nodes and freedoms. Also known as a monset in Elfini.

10.9.3 Examples

```
extract { a = AEKHH{ rows=(1 to 10 by 2), col=(1 to 10 by 2) } }
```

will create a (5,5) matrix consisting of row and columns 1, 3, 5, 7, and 9 of AEKHH; A simpler way is to use the rset option:

```
extract { a = AEKHH{ rset=(1 to 10 by 2) }
```

```
extract { d = AEKHH{diag} }
```

will create a matrix of all the diagonals of AEKHH.

```
extract { a = "modes,set=33"{rows=(2021/tx, 2021/2, 3021/1, 3021/ty)} }
```

will create a matrix with four rows from a matrix named modes,set=33 (note the use of quotes to protect the comma) corresponding to nodes 2031 and 3021 freedoms tx and ty. The node and freedom information is from set 33.

```
extract { a = gaf{rfi=0.02, diag=(191 to 196)} }
```

will create a (6,1) matrix of the diagonals of matrix gaf evaluated at rfi=0.02.

```
extract {
   a = identity{col=3, size=10}
   b = I{col=4, size=10}
   c = I{col=6, size = 10}
}
```

will create three (10,1) matrices which are zero except for a 1 in row 3, 4, and 6, respectively. Note the placement of the column option inside curly braces following the matrix name, and the common option size outside the braces.

10.10 stab

Solves the aeroelastic stability (flutter) equation producing continuous curves of standard and user-defined parameters.

10.10.1 Syntax

stab { *options* }

10.10.2 Summary

A wide variety of solutions are possible depending on the choice of active (independent) parameters. Solutions are generally classified as either neutral-stability or parameter variation. The type of solution is controlled by the choice of active (independent) parameters.

Neutral-stability curves have for actives a parameter which increases dynamic pressure, e.g. vtas, veas, or dynamic pressure itself; frequency; and a parameter that relates to growth or decay of oscillations: sigma, growth rate, or sdamp.

Parameter variations start from a point on a neutral-stability curve and trace a curve varying another parameter.

Chapter 5 lists predefined parameters and explains how to define new parameters; section 7.1 discusses the flutter equation:

$$\boldsymbol{Dq} = \left[s^2\boldsymbol{M} + s\boldsymbol{G} + s\boldsymbol{V} + (1+id)\boldsymbol{K} - q\boldsymbol{Q}(p,M) + \boldsymbol{T}\right]\boldsymbol{q} = \boldsymbol{0} \qquad (10.1)$$

10.10.3 Options

See §10.1 for the general syntax of options.

Options are categorized as

- Identifying The Solution
- Equation Definition Options
- Which Aeroelastic Modes to Track
- Output

Identifying The Solution

id = *string*

Analysis identifier which is used to identify the solution results for subsequent parameter variations (see the source option) and for the default plotfile name. It may be arbitrarily long but should not contain spaces.

Default: combination of the active-parameter names, e.g. vtas-freq-growth

source = *string*

An id option that was used to tag the output of a previous flutter solution. Including the source option has two main effects on this run: equation definition options (matrices, parameters, rset, and eset, but **not** active or aeromod) will be implicitly used, and the solutions from the previous run will be used as start points for tracing aeroelastic modes in this run.

Equation Definition Options: Parameters

Exactly three active parameters must be specified (unless the optimize option is included) and enough fixed parameters to allow all remaining parameters to be derived from these.

In some rare situations it is necessary to declare parameters derived, meaning the program is to find suitable equations in terms of the other parameters. The derived option does this, and is necessary only when the source option is included, because with the source option the default state (§5.3) for all parameters is the state used in the source run; the default state can be changed with the active or derived options or by specifying a value or values for the parameter (which sets the state to fixed). An example of when the derived option is necessary is when a stab command uses a fixed value for a parameter, followed by another stab command including the source option but in the second command the parameter must not be fixed but derived from the other parameters:

```
stab { id=pk, mach=0.8, ...}
stab { id=pv, source=pk, derived=mach, ...}
```

Without the derived option `mach` would have a fixed value of 0.8 in the second command.

active = *stringList*

A list of parameters defined to be active in the solution. These may be standard or user defined parameters.

Default: In general three parameters must be defined as active; the only exception is if the optimize parameter is included.

derived = *stringList*

A list of parameters defined to be derived in the solution. These may be standard or user defined parameters. See the discussion above for an explanation of when this option is necessary.

optimize = *string*

The name of a parameter to be optimized using the continuation optimization technique (§7.1.3). When this option is included the number of active parameters can be any number greater than or equal to three.

parameter-defn

Parameters are declared *fixed* by including at least the parameter name, an equals sign, and one or more values (multiple values are enclosed in parentheses and separated by commas). Parameters may also be given equations as detailed in §5.4 which make them *derived*.

Equation Definition Options: Matrices

The matrices used in equation10.1 are specified with the following options where string is the name of an existing FLAPS matrix. Most are optional; the exceptions are mass and stif.

mass = *string*

the name of the matrix to be used as the mass matrix $\boldsymbol{M}$

Default: fatal error

stif = *string*

the name of the matrix to be used as the stiffness matrix $\boldsymbol{K}$

Default: fatal error

vdamp = *string*

the name of the matrix to be used as the viscous damping matrix $\boldsymbol{V}$

gyro = *string*

the name of the matrix to be used as the gyroscopic matrix $\boldsymbol{G}$

gaf = *string*

the name of the unsteady aerodynamics matrix $\boldsymbol{Q}$

aeromod = *string*

modification of the generalized aerodynamic matrix by either the NASTRAN method (§3.5.1) (aeromod=nastran) or the g-method (§3.5.2): aeromod=gmethod.

controls = *string*

name of the control-law matrix $\boldsymbol{T}$

Rows and columns of the matrices may be removed prior to solving the equations by specifying rows and columns to remove (eset) or rows and columns to retain (rset). All references to generalized-coordinates, for example in printed summaries or output-transformations, use the numbering of the full set of equations (prior to extracting rows and columns). User-defined subroutines must behave as though no rows and columns had been removed.

eset = *intList*

A list of one or more rows and columns to be removed from all matrices prior to each solution.

rset = *intList*

A list of one or more degrees-of-freedom to be included in the solution. Rows and columns of all matrices will be extracted in the same order as specified and assembled into the dynamic matrix.

Default: all degrees-of-freedom are included

Which Aeroelastic Modes to Track

The method for choosing aeroelastic modes to track depends on whether this is a neutral-stability or parameter-variation run. The startregion option is the most useful way to select which modes to track. For neutral-stability runs modes can be selected by frequency. Parameter-variation modes can be selected by any parameter that narrows the range of modes; for example many parameter variation modes can be limited by vtas. A second method depends on knowing the ordering of low-speed aeroelastic modes by frequency; the modes options specified which low-speed aeroelastic modes ordered by frequency are to be tracked.

startregion { *options* }

a list of active parameters with limits or values specifying the region in which to search for start points. Limits are specified in the usual way (§5.1), for example

```
stab { ..., startregion{freq[0:10]}, ...}
```

Specifying a value for a parameter is the same as specifying both limits the same; for example

```
stab { ..., startregion{veas[10:10]}, ...}
```

is equivalent to

```
stab { ..., startregion{veas=10}, ...}
```

It is important to note that limits and values in the startregion option only affect the start points; they have no influence on the aeroelastic modes beyond the start points.

Default: the active parameter limits

modes = *intList*

a list of the aeroelastic modes to be tracked. The aeroelastic modes are numbered according to increasing natural vibration frequency.

Default: all modes in the startregion are tracked

Output

Output from stab consists of printed output, plot files, and data stored in the FLAPS database.

Printed output is controlled by the print, target, goal, and energy options. The print option specifies which parameters are printed; the default is to print the active parameters, multiple-valued fixed parameters, and a summary of either the generalized-coordinates or energies. Most of the output is self-explanatory; section 10.4.2 explains the generalized-coordinate (or energy) summary.

Plotted output is written to one or more ESA-formatted [34] files. Which parameters are included is determined by the plot option (default is all the parameters), and the file names are controlled by the plotfile, append, continuecuts, and nosplit options.

The continuecuts and nosplit options are mutually exclusive and need some explanation. Multiple-valued parameters are often referred to as *parameter cuts* (§7.3.1); the default behavior in stab is to output a different plotfile for each value of multiple-valued parameters. Unique plotfile names are created from the name specified with the plotfile option by adding an index. For example a stab run with three values of a parameter p1:

```
stab { id=pk, p1=(22.3,22.4,22.5), ... }
```

would produce plotfiles named `pk.1.esa, pk.2.esa` and `pk.3.esa`.

It is sometimes convenient to have multiple stab commands create a series of plot files with continuous cut numbers For example the two stab commands

```
stab { id=pk, p1=(22.3,22.4,22.5), plotfile=myplotfile, ... }
stab { id=pk2, p1=(32.3,32.4,32.5), plotfile=myplotfile, continuecuts, ... }
```

would produce plotfiles named `pk.1.esa, pk.2.esa` and `pk.3.esa` (from the first command) and `pk.4.esa, pk.5.esa` and `pk.6.esa` from the second command. The continue-cuts option changes the numbering in the second stab command and prevents the first three files from being overwritten. Alternatively, solutions from both stab commands could be put into one file by including the nosplit and append options:

```
stab { id=pk, p1=(22.3,22.4,22.5), plotfile=myplotfile, nosplit, ... }
stab { id=pk2, p1=(32.3,32.4,32.5), plotfile=myplotfile, nosplit, append, ... }
```

The target option specifies certain criteria which if met prints the results at that point. A related option, goal, specifies parameter values which, if **not** met result in a warning message.

Output transformations (§5.7) are parameters defined with equations containing references to the generalized-coordinates. For example

```
stab { ... n7tzvel(Node 7 TZ Vel) = s*n7tz*gc, ... }
```

where n7tz is the row of the modes matrix corresponding to the tz freedom of node 7. The new parameter n7tzvel will be included in the plot file by default.

print = *stringList*

a list of parameter names to be included in the printed output. Certain keywords in the list have special meanings unless there is also a parameter by that name:

full	print a one-line summary for each solution point on every solution curve; the default is to print the first and last points and any targets found.
nogc	do not print a summary (§10.4.2) of the generalized-coordinates; the default is to include the summary.
matrices	print all the matrices used in the flutter equation.

target { *options* }

specifies one or more parameter values where results are to be printed and saved; additional parameters with limits may be included to narrow the window where targets are sought. For example,

```
stab { ... target{ growth=(0,0.03), vtas[200:300] }, ...
```

will print solutions between 200 and 300 knots tas where growth is either zero or 0.03.

goal { *options* }

one or more parameters who's minimum or maximum is expected to be met. For example,

```
stab { ... goal{ vtas=max }, ...
```

will print a warning if the maximum vtas is not reached for any reason by any aeroelastic modes which are tracked. Reasons for not reaching a goal include reaching a limit in one of the active parameters before the goal is reached, or a curve runs into numerical problems.

plot = *stringList*

a list of parameters to be included in the plot file. See section 5.5. for a list of predefined parameters.

Default: all parameters.

plotfile = *string*

name of the file to receive the ESA-formatted plot data.

Default: the solution identifier specified with the id option with a `.esa` extension

append

If there is an existing file with the plotfile name the results are appended to the file, rather than the default behavior which is to delete the contents before writing the new results.

Default: the contents of existing plotfiles are deleted

nosplit

Put the solutions for all parameter cuts into the same plot file (see discussion above).

Default: results are written to plot files with multiple-valued parameter indices included in the file name.

continuecuts

continue parameter cut indices from the highest index found in files with the same base plotfile name on the current directory (see discussion above).

Default: if there are multiple-valued parameters declared, the parameter value indices are included in the plotfiles names starting with 1.

energy

print summaries of the generalized-coordinate energies instead of the g.c. magnitudes any place where g.c. magnitudes are normally printed.

Miscellaneous Options

nrbm = *int*

the number of rigid-body modes in the model. Including this option causes the first nrbm rows and column of the stiffness to be zeroed. This option is to compensate for generalized stiffness matrices which should have zeros in the first nrbm rows and columns, but due to inaccuracies in eigensolutions have non-zeros in those rows and columns, sometimes large enough to change a flutter solution.

minstep = *int*

the minimum number of steps to take when tracking each mode. This option is useful for creating smoother curves for plotting.

Default: 50

maxstep = *int*

the maximum number of steps to take when tracking each mode.

Default: 1000

10.11 gyro

Creates a gyro matrix suitable for use in flutter or response analyses. The output matrix will be reduced to generalized coordinates if the modal or modes options are included; otherwise it will be in nodal degrees-of-freedom associated with the specified set, ss, or model. **The units of the generalized coordinates associated with the output gyro matrix are assumed to be radians; It is the user's responsibility to ensure that the output matrix is scaled properly if this is not the case.**

10.11.1 Syntax

gyro { *options* }

10.11.2 Options

See §10.1 for the general syntax of options.

set = *int*

Identifier for the nodal data; ATLAS uses integers Elfini uses strings (monset), NASTRAN uses strings.

Default: fatal error

modal

The output matrix is transformed to the generalized-coordinates associated with the modes matrix for the specified set by doing a triple-product.

Default: the output matrix is in nodal degrees-of-freedom unless the modes option is included.

modes = *string*

The name of the matrix to use in transforming the output matrix to generalized-coordinates with a triple-product. This is an alternative to the modal option when a modes matrix other than the one associated with the specified set is to be used.

Default: the output matrix is in nodal degrees-of-freedom unless the modal option is included.

o = *string*

Name to be given to the output matrix.

Default: gyro

node = *int* {*options*}

The number of the node where the gyro forces are to act. This option may be repeated as many times as necessary to define all the spinning parts in the model. At least one node must be included. Options following the node number specify the direction and value of the rotation rate and rotary inertia:

inertia = *float*

Rotational inertia in lb_f-*in*-sec^2. Multiply inertia in lb_m-in^2 by the conversion factor $G = \frac{lb_m in}{lb_f sec^2}$. (§2.5)

Default: 1 lb_f-*in*-sec^2

orient = *floatList*

Three floating-point numbers which specify the direction of the spin vector relative to the node. The direction determines the orientation of the spin axis and the direction of the rotation according to the right-hand-rule.

Default: fatal error

spin = *float*

Spin rate (magnitude of the spin vector). Negative values indicate the rotation direction is opposite that specified by the orient option.

Default: 1.0 rad/sec

10.11.3 Output

Output from this command consists of a skew-symmetric matrix which may be used in the stab, fresp, or tresp commands by including the gyro option.

10.11.4 Examples

The following FLAPS command will create a gyro matrix named "G" in modal dof associated with ATLAS set 2, reduced to generalized coordinates associated with ATLAS matrix `"modes,set=2"`. The spin vector is in the negative y direction and it is spinning at 5 rad/sec.

```
gyro {
   set=2, modal, o=G,
   node=112{orient=(0,1.0,0), spin=-5, inertia=3300}
}
```

10.12 import

Reads files in various formats and creates FLAPS matrices from the contents.

10.12.1 Syntax

import { *path* }

10.12.2 Options

See §10.1 for the general syntax of options.

Options are categorized as

- General Options
- ESA Options
- NASTRAN/BAP Options

General Options

i = *string*

The path of the file to be imported; not relevant for BAP databases (see dbmaster option).

format = *string*

The format of the data in the file. It is normally not necessary to specify the format - it can be determined from the file extension if the default extension is used, or by examining the data. Valid formats are

apex FLAPS ASCII format created by the ATLAS export module. No default extension.

output4 NASTRAN or BAP OUTPUT4 formatted ASCII file [37]. Default extension: **.op4**.

dmig NASTRAN or BAP DMIG formatted ASCII file [37]. Default extension: **.dmig**.

bap BAP database files; it is unnecessary to specify this format option because the dbmaster option (required for reading BAP database files) makes it redundant. See the NASTRAN/BAP options below.

elfini Elfini Neutral file (ASCII) Default extension: **.nf**.

esa The ASCII form of Boeing Engineering Scientific Data (ESD) [15]. The binary form of ESD, known as ESB, is commonly used for plotting data in pegasus ([34]). Default extension: **.esa**.

matlab A MAT-file from Matlab or Simulink. These files are binary but unlike FLAPS binary savefiles they are not in general portable across platforms [23]. Default extension: **.mat**.

matrix market An ASCII format widely used in the numerical analysis community for distributing test matrices. Used for printing large matrices in FLAPS and viewing with matview (§10.6) [24]. Default extension: **.mm**.

universal An ASCII format originally from SDRC, widely used to transfer data to laboratory test software. Only formats 15 (nodal), 55 (modes) and 82 (connectivity) are allowed (format 164 (units) is ignored). The formats are documented at [25]. Default extension: **.uf**.

Default: the format will be determined from the file extension or by looking at the data.

ESA Options

runid = *stringList*

a list of runids or regular-expressions (§I) to be imported; **only applicable when** format=esa.

param = *stringList*

a list of parameters or regular-expressions (§I) to be imported; **only applicable when** format=esa.

NASTRAN/BAP Options

Importing data from NASTRAN or BAP databases is more complicated than simply specifying a file name. Access to these databases is through a toolkit provided by the vendor (MSC); this toolkit requires that NASTRAN be run even though all we want to do is read data. For this reason it is only possible to import from NASTRAN/BAP databases on platforms which also have NASTRAN/BAP and the necessary licenses. It also means that the options necessary to import are arcane.

dbmaster = *string*

path of a NASTRAN master database

Default: fatal error if the master database is not specified

fetch = *stringList*

a list of the matrix names to fetch from the NASTRAN database. The matrices are stored in the FLAPS database; to create an FLAPS savefile use the save command (§10.20).

Default: not much point in using the import command unless this option is included :-)

dball = *string*

path name of the (optional) NASTRAN DBALL database

adb = *string*

path of a NASTRAN adb database.

sdb = *string*

path of a NASTRAN sdb database.

bapversion = *string*

either a number specifying the version of BAP to use or the path of the BAP executable to use.

dbproject = *string*

the project identifier

Default: any project

dbversion = *string*

the database version identifier

Default: any version

buffersize = *int*

the buffersize of the databases; necessary if the databases use a buffersize other than the BAP default (8192). Why NASTRAN/BAP requires this is a mystery.

10.12.3 Command-line Usage

import can also be used on the command-line to read NASTRAN/BAP databases and create an FLAPS savefile. Although the import command can be used directly on the command-line if you have it in your path, it is more convenient to call it from the apex script:

```
$ apex import ...
```

The only command-line options are a list of the matrix names (see example below). All other options are specified with environment variables which must be set prior to executing import.

`NASTRAN_DB_MASTER` (required) path to a NASTRAN/BAP master database.

`NASTRAN_DB_ALL` path to a NASTRAN/BAP DBALL database if present.

`BAP_VERS` path to BAP executables.

`NASTRAN_BUFFSIZE` buffer size - required for databases that have a buffer size that is larger than the default value (8192).

`DEBUG` (optional) debug level from 1 to 4.

Example

The following commands (which you would probably want to put in a script) extracts several matrices from BAP databases and creates three FLAPS savefiles:

```
export NASTRAN_DB_MASTER=/787-8/Prel/R803/BAP/Merge/R803_ips_sdb.MASTER
export BAP_VERS=/sd/core/bap/bap2d13
export NASTRAN_BUFFSIZE=32769

apex import AEMHH,MASSID=0 AEKHH
mv savefile R803_model.sf

export NASTRAN_DB_MASTER=/787-8/Prel/R803/BAP/Merge/R803_ips_adbm.MASTER

apex import modes,set=101 freqs
mv savefile R803_modes.sf

export NASTRAN_DB_MASTER=/787-8/Prel/R803/BAP/Merge/R803_ips_adbd.MASTER

apex import AEQHH
mv savefile R803_theo_gaf.sf
```

The example runs import 3 times. Prior to each execution the NASTRAN_DB_MASTER environment variable is modified for a new database. The first run extracts generalize stiffness (AEKHH) and generalized mass (AEMHH) matrices. MASSID=0 is the qualifier/value pair that specifies the base mass case for an SDBmgr database.

The second import run extracts modes and frequencies for a set of nodes defined in a NASTRAN case control SET entry. "modes,set=101" refers to three matrices that are created from data in the the ADBmgr_m database. import returns matrices "freedoms,set=101", "modes,set=101", and "nodes,set=101". In addition to the modes a vector of frequencies is extracted from the AELAMA table and store in "freqs".

The third run extracts the generalized airforces from the AEQHH datablock. This example will extract all of the unique matrices "AEQHH,mach=x,rfi=y" where x and y are the Mach numbers and reduced frequencies. The attribute names Mach and rfi are different from the qualifier names IMACHNO and IKBAR from the NASTRAN/BAP database. These were change because Mach and rfi are standard names used in FLAPS.

Matrix Names

In NASTRAN matrices and tables are stored in the database in datablocks. Each datablock has a "path" of qualifier/value pairs that are used to locate a unique matrix or table. In general any single matrix in a NASTRAN database can be extracted by specifying its datablock name followed by a series of qualifiers that uniquely defines the matrix. For example:

```
apex import AEMHH,MASSID=100
```

extracts all matrices in database (BAP) with a DATABLOCK name of AEMHH and qualifier MASSID equal to 100. This example specifies one matrix so only one matrix will be saved in the FLAPS savefile format.

```
apex import AEMHH
```

Does not uniquely specify a DATABLOCK name, so the number of matrices returned will equal the number of unique MASSID values.

Debugging

import launches NASTRAN through the toolkit when accessing a database. For that reason debugging problems may be problematic. The toolkit itself has very limited error reporting, that is of little use to a import user. To aid in debugging problems the user can set the DEBUG flag to get additional information. The DEBUG flag can be set to values from 1 to 4, however setting values above 2 will generate a considerable amount of output.

If import is run with the DEBUG environment variable set two things happen the output written to the screen is greatly increased and some temporary files are left behind. The temporary files begin with `DBATTACH`, the most important of which is `DBATTACH_.....f06` output. This file contains the NASTRAN output of the toolkit job that connects to the database specified by the `NASTRAN_DB_MASTER` environment variable. If modes extraction is requested this file also contains output from a DMAP program that partitions the "g-set" modes matrix.

Known Bugs

When extracting generalized air force matrices from a BAP database, all AEQHH matrices are extracted. This means that if you have an ADBD database with GAF matrices for two Mach numbers (x and y) both Mach numbers will be extracted even if one (IMACHNO=x) is specified.

If any nodes have local coordinate definitions there displacements will be output in the local reference frame. This issue is being worked.

10.13 merge

Merge one or more FLAPS matrices, creating a new FLAPS matrix.

10.13.1 Syntax

merge { *options* }

10.13.2 Options

See §10.1 for the general syntax of options.

Options to merge consist of the output matrix name, and a set of input matrix names. The input matrices are stacked rowwise or columnwise according to whether the input matrix list is specified using the row or col option. You may **not** include both row and col in the same merge statement.

Not all datatypes can be merged; real and complex matrices may be merged either rowwise or columnwise. Nodes, freedoms, and subsets may be merged with the row command but not the col command.

output-name { *options* }

Output matrix name. The merge options are:

rows = *stringList*

Matrix names which are to be merged as block rows of the output matrix. A matrix name `0` means to insert a zero row at that position.

Default: none

col = *stringList*

Matrix names which are to be merged as block columns of the output matrix. A matrix name `0` means to insert a zero column at that position.

Default: none

10.13.3 Examples

To merge the (1,193) matrix tya, the (1,193) matrix rza, and the (2,193) matrix rxa to form a (4,193) matrix psi:

```
merge {psi{rows=(tya, rza, rxa, rxa)}}
```

10.14 output

Set options controlling the output from subsequent commands.

10.14.1 Syntax

output { *options* }

10.14.2 Options

See §10.1 for the general syntax of options.

out = *string*

file name where standard output is to be redirected

Default: none

err = *string*

file name where standard error is to be redirected

Default: none

environment = *string*

environment variable to be set for subsequent commands. The right-hand-side must be a quoted string. Double quotes and single quotes behave differently in the way that environment variables are handled: if the right-hand-side is enclosed in double quotes any environment variables in the quotes are expanded when the control program starts. Single quotes delay the expansion until the output command is executed. This would make a difference if an environment variable is changed in the control program and later referenced; see Examples.

Default: none

pagewidth = *int*

maximum number of characters per line of printed output. Setting the environment variable PAGEWIDTH has the same effect. Most printout from FLAPS commands is limited to approximately 130 character lines, so increasing the pagewidth will mostly have no effect; two places where it will are in the tracking summary in stab where the number of generalized coordinates printed is limited by the pagewidth. The other effect increased pagewidth will have is in the output from the print command, which prints matrices to a .mm file ([24]) for visualization in MatView if the matrix is too large for printed output; increasing the pagewidth enough will result in the matrix being printed to the output file instead of a .mm file.

Default: 130

timer = *int*

change the amount of timing information. Some FLAPS commands write a table of times spent for various parts of the command to stderr; setting the level of print to zero means the table will not be written. Setting the level to 1, 2, or 3 gives more detailed timing information.

Default: 1

10.14.3 Examples

To write subsequent output to a file named junk on the home directory, and to discard standard error:

```
output { out=$HOME/junk, err=/dev/null }
```

HOME is an environment variable which is set automatically to the user's home directory. The file `/dev/null` is the standard Unix trashcan (anything written to this file is discarded).

As an example of setting environment variables from within an FLAPS control program consider

```
export Lnv=lnv
apex << @
   output { env="Lnv=${Lnv}a" }
     ...
   output { env='Lnv=${Lnv}b' }
     ...
   output { env="Lnv=${Lnv}c" }
```

After the first output statement environment variable Lnv is `lnva`, after the second it is `lnvab`, but after the third it is `lnvc` due to the use of double quotes instead of single

quotes; double quotes mean the environment variable is replaced when the control program is read by FLAPS, whereas an environment variable in single quotes does not get replaced until it is used - in this case when the output statement is executed.

Environment variables can be used to streamline control programs; for example to create and use an identifier for a job

```
output { env="id=pk85" }
  ...
stab { id = $id, plotfile = $id.esa, ... }
  ...
vis { id = $id, o = $id.uf, ... }
```

10.15 param

The param command is used to create matrices that are functions of arbitrary parameters.

10.15.1 Syntax

param { *options* }

10.15.2 Summary

Several types of parameterizations are available; in roughly decreasing order of generality they are

user subroutine associates a user-written subroutine with a new or existing matrix; the matrix is passed to the subroutine where it can be arbitrarily modified

matrix element assign an equation to a matrix element

interpolation general interpolation in any number of variables

rational function approximation approximate a set of generalized aerodynamic matrices as a function of complex reduced frequency

branch mode frequencies make a diagonal element of a stiffness matrix a function of a function of a so-called branch-mode frequency.

structural damping make specified rows and columns of a stiffness matrix a function of a parameter representing the structural damping on those elements.

ABCD control-laws create a matrix representing a control-law using data from Matlab/Simulink

Some are for specific types of matrices. For example, branch mode frequencies and structural damping parameterizations are only for stiffness matrices, RFA parameterization is only for unsteady aerodynamics matrices, and ABCD parameterizations are only for representing controls equations.

Multiple parameterizations can sometimes be associated with a matrix. For example, a stiffness matrix could be parameterized with branch mode frequencies, structural damping, matrix element equations, interpolation, and a user-subroutine, all acting on the same matrix. The param processor arranges for the parameterizations to take place in the following order: first the interpolation, then the diagonals associated with the frequencies, structural damping, and matrix-element equations would be applied, then the user-subroutine would be called with the resulting matrix. Not all combinations make sense; interpolation and RFA cannot be used on the same matrix, nor can RFA

and ABCD. User-subroutines are the most general type of parameterization, and can be used with virtually any matrix and any other parameterization.

10.15.3 Options

See §10.1 for the general syntax of options.

Options are categorized as

- General Options
- Plot Options
- Matrix element
- Branch mode frequency parameterization
- Structural damping
- Interpolation
- Rational function approximation
- ABCD control-laws
- User-subroutine parameterizations

General Options

Input matrices may be specified either by just including the name or as the right-hand-side of the i option. The right-hand-side of the i option may also be list of matrix names enclosed in parentheses. So, for example

```
param { genforce }
```

and

```
param { i = genforce }
```

are equivalent. Matrix attributes (§2.3) may be left off; if they are, all matrices with the missing attributes will be used. For example,

```
param { i = genforce }
```

is equivalent to (assuming these are all the genforce matrices in the database):

```
param {
  "genforce,aerocase=3,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=3,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
  "genforce,aerocase=4,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=4,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
  "genforce,aerocase=5,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=5,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
  "genforce,aerocase=6,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=6,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
  "genforce,aerocase=7,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=7,aerocond=2,rfr=0,rfi=0.04,mach=0.8"
}
```

whereas

```
param {
  "genforce,aerocase=3,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=4,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=5,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=6,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
  "genforce,aerocase=7,aerocond=2,rfr=0,rfi=0.02,mach=0.8"
}
```

is equivalent to

```
param { i = "genforce,rfi=0.02" }
```

Parameters are associated with matrices in two ways: by the attributes which are part of the name or by parameter definitions enclosed in curly braces following the name. In the example above, the statement

```
param { i=genforce, o=gaf }
```

will interpolate with respect to rfi, the only floating-point attribute that varies among the input matrices. As an example of the second type of parameter association, the statement

```
param {
   i = (gm1{fuel=0},gm2{fuel=25},gm3{fuel=50},gm4{fuel=75},gm5{fuel=100})
   o = Mass
}
```

interpolates 5 mass matrices with respect to a new parameter called `fuel`, creating a new matrix called `Mass`. The matrix does not have to have been parameterized with respect to the parameter in curly braces; if it has it will be evaluated at the parameter value, otherwise it will simply be taken as the matrix at that value of the parameter for interpolation with respect to that parameter. So, for example the statement

```
param {
   i = Mass{fuel=0}
   i = Mass{fuel=100}
   o = LMass
}
```

will create a new matrix `LMass` which is a linear interpolation between `gm1` and `gm5`.

i = *stringList*

The name of the input matrix or a list of names enclosed in parentheses.

Default: None.

o = *string*

The name given to the output matrix.

Default: Name of the input matrix if there is only one, fatal error otherwise.

parameter-defn { *options* }

parameter-defn is a parameter definition as detailed in section 5.1. The options link the parameter to the input matrix as detailed below.

datatype = *string*

The data type to be given to the output matrix. Valid types are real and complex.

Default: The output matrix usually has the same datatype as the input matrix unless the parameterization requires casting the matrix to a different datatype; for example applying a structural damping parameterization to a (real) stiffness matrix requires the output matrix be complex.

Plot Options

It is sometimes useful to plot elements of parameterized matrices to check the parameterization.

plot { *options* }

Create plot files of selected elements of the output matrix versus one or two parameters. The active option specifies the parameter(s) to be varied and their limits; other parameters may be set constant by listing them with with a value on the right-hand-side. Declaring one active parameter means an ESA-formatted plot file will be created; two active parameters results in a Universal file which can be visualized using amvis. The data for each element plotted is written to a separate file with a name like `output[row,col].esa` where `output` is the name given to the output matrix, and `row` and `col` are the row and column numbers. The options are:

active = *stringList*

a list of one or two parameters to be varied for plotting; one active parameter produces an ESA-formatted plot file, two actives produces a Universal file. For example

```
plot{ active=vtas, ... }
```

Default: all parameters the output matrix is a function of, except parameters declared fixed.

parameter-defn

Parameters are declared *fixed* by including at least the parameter name, an equals sign, and one or more values (multiple values are enclosed in parentheses and separated by commas). Parameters may also be given equations as detailed in §5.4 which make them *derived*.

diag

plot all the diagonal elements

diag = *intList*

a list of diagonal elements, e.g.

```
plot{diag=(1 to 30 by 4)}
```

row = *intList*

a list of rows to plot; if the col option is also included, the elements will also be limited to those columns.

col = *intList*

a list of columns to plot; if the row option is also included, the elements will also be limited to those rows.

[*i*,*j*] = *string*

Plot the element in row *i*, column *j*. This option may be repeated as many times as necessary; e.g.

```
plot{ [1,3], [3,1], [4,6] }
```

Parameterizing Matrix Elements

Individual elements of the output matrix may be given equations by specifying the row and column numbers in square brackets on the left and the equation on the right side of the equal sign:

[*row*,*col*] = *string*

the row and column number enclosed in square brackets on the left side of the equal sign, and a parameter equation (as detailed in section 5.4) on the right side.

For example

```
[27,25] = 3.2*lnvb*log10(alt)*sqrt(Mass[27,27])
```

sets the (27,25) term of the output matrix to an equation involving a user-defined parameter (lnvb), standard parameter alt, and the (27,27) element of a matrix called Mass.

Branch Mode Frequencies

If the input matrix is a stiffness matrix associated with a structure in which some generalized coordinates exhibit the branch-mode property (§6.3.1 and appendix H), it can be parameterized by the frequencies of specified generalized coordinates by including the bmfgc and mass options; a parameter definition may be associated with the generalized coordinate by following it enclosed in curly braces:

bmfgc = *int* {*parameter-defn*}

> A generalized-coordinate number There should be no coupling between this generalized coordinate and others in the stiffness matrix. Following the coordinate number may be a *parameter-defn* enclosed in curly braces. The values of this parameter are used to set the diagonal element of the stiffness matrix according to the single-dof formula $\omega = \sqrt{\frac{\boldsymbol{K}_{ii}}{\boldsymbol{M}_{ii}}}$

mass = *string*

> Name of the generalized mass matrix associated with the input generalized stiffness matrix.
>
> Default: the last specified mass matrix; fatal error if none

Structural Damping Parameterizations

If the input matrix is a generalized stiffness matrix it can be parameterized by applying structural damping to one or more generalized coordinates; structural damping parameterization uses a syntax similar to branch mode frequency parameterization: the word sdgc on the left-hand side of the equals sign, a list of one or more generalized-coordinate numbers follow by a parameter definition (5.1) enclosed in curly braces.

sdgc = *intList* {*parameter-defn*}

> A list (§10.1) of one or more generalized-coordinate numbers; the structural damping is applied by replacing each element in these rows with the real part of the element times the complex number $(1+id_k)$ where d_k is the value of the parameter. *parameter-defn* is a parameter definition as detailed in section 5.1. This multiplication takes place prior to multiplying the entire matrix by $(1+id)$ where d is the value of the structural damping parameter sdamp (§5.5). See the caution regarding the use of both in §6.3.2.

Interpolation

If there is more than one input matrix, and the beta option is not included, the matrices are fitted with a *tensor product spline* (see [28] chapter 4) which is a spline interpolation with any number of independent variables. For example, unsteady aerodynamic matrices which are functions of reduced frequency and Mach number are interpolated with respect to both reduced frequency and Mach number (a surface). The degree of the splines can be specified, and defaults to 3: cubic splines.

If in the list of input matrices some of the names appear twice, and there is only one independent parameter, a separate spline is fitted on either side of the two matrices. This allows for matrices which have slope discontinuities with respect to the interpolation parameter.

Parameters are associated with an input matrix in two ways: as attributes which are part of the name (e.g. `"genforce,rfi=0.1,mach=0.4"`), or as explicit parameter definitions enclosed in curly braces following the name, for example

```
i = gm1{fuel=0}
```

degree = *int*

The spline degree for the interpolation.

Default: 3 (cubic spline)

degree { *options* }

The spline degree for the interpolation may be specified for each parameter by giving the degree for any of the interpolation parameters; for example to interpolate a set of aero matrices with respect to rfi and mach with linear interpolation on mach and cubic interpolation on rfi:

```
param { ..., degree{mach=1}, ... }
```

because rfi is not included its degree is 3 (the default).

Approximation

Approximation of a set of unsteady aerodynamic matrices by a rational-function approximation (appendix E) is triggered by including the beta option. The input matrices must be functions of rfi (§5.5.1)

beta = *floatList*

Values of β used in the rational-function approximation.

ABCD Control-Laws

The abcd option specifies a file containing a description of a control-law in the ABCD form (see appendix C) produced by Matlab/Simulink [23]. The format of this file is detailed in appendix D.

abcd = *string* {*options*}

path of the file containing the ABCD matrices followed by options. Output from the control-law is determined by one (and only one) of the options ke or e. It is important to be aware of the difference between the two. If the stiffness matrix **K** has been parameterized in a way that the product **KE** is not constant, then it is necessary to use the e option, so that the product is formed using the modified stiffness matrix.

psi = *string*

name of the control-law input matrix. The number of rows in this matrix must match the number of control-law inputs in the ABCD file. The extract command (§10.9) can be used to create this matrix; see Examples.

Default: psi

ke = *string*

name of the control-law output matrix. The number of columns in this matrix must match the number of control-law outputs in the ABCD file. This option is an alternative to the e option. It is important to be aware of the distinction between this option and the e option as discussed above. The extract, merge, and alge commands (§10.9, 10.13 and 10.5) can be used to create this matrix; see Examples.

Default: fatal error unless the e option is included

e = *string*

name of a matrix (**E** in appendix C) which relates the control-law output to the generalized coordinates. The number of columns in this matrix must match the number of control-law outputs in the ABCD file. It is important to be aware of the distinction between this option and the ke option as discussed above.

Default: fatal error unless the ke option is included

igain { *parameter-defn, ...* }

list of parameter definitions for the control-law input gains; either one definition for all inputs or one for each input (row of the psi matrix).

Default: igain=1

iphase { *parameter-defn, ...* }

list of parameter definitions for the control-law input phases; either one definition for all inputs or one for each input (row of the psi matrix).

Default: iphase=0

ogain { *parameter-defn, ...* }

list of parameter definitions for the control-law output gains; either one definition for all outputs or one for each output (column of the KE matrix).

Default: ogain=1

ophase { *parameter-defn, ...* }

list of parameter definitions for the control-law output phases; either one definition for all outputs or one for each output (column of the KE matrix).

Default: ophase=0

User-Subroutine Parameterization Options

User-subroutine parameterization can be used to modify an existing matrix or to create a new matrix. If one (and only one) input matrix is specified it will be passed to the subroutine whenever the matrix is to be evaluated; otherwise a matrix of zeros will be passed to the subroutine. Section 6.4 has guidelines for writing a subroutine.

code = *string*

> Name of a file (or a block in the FLAPS submit file) containing a Fortran subroutine which modifies the input matrix. The datatype of the output matrix is determined by the declaration of the matrix in the subroutine; for example the code
>
> ```
> subroutine usrtf (n, k)
> real k(n,n)
>    ...
> ```
>
> will result in a real output matrix, and
>
> ```
> subroutine usrtf (n, t)
> integer n
> complex t(n,n)
>    ...
> ```
>
> will produce a complex output matrix.

The dimensions of the output matrix are determined by the dimensions of the input matrix and optionally extra rows and columns specified with the `extra` option, or if there is no input matrix by the `size` option.

size = *stringList*

> The dimensions of the output matrix; the list can of one or two strings: the first specifies the row dimension and the second the column dimension. Specifying only one string yields a square matrix. The strings can be either integers or the name of an existing matrix. If a matrix name is specified the corresponding row or column dimension will be taken from the matrix. For example, if `Sensor` is an existing (3,5) matrix and `AEKHH` is an existing (5,5) matrix
>
> ```
> size = (3,5)
> size = (Sensor,5)
> size = (3, Sensor)
> size = (Sensor, AEKHH)
> ```
>
> all produce the same size matrix, and

```
size = (5,5)
size = 5
size = AEKHH
```

all produce a (5,5) matrix.

extra = *int*

The number of extra degrees of freedom to include in the output matrix. Extra rows and columns are appended to the bottom and right of existing matrices.

10.15.4 Examples

The following FLAPS command examples are taken from sample problem stab1. The first command creates a interpolation matrix named 'Mass' that has the parameter 'cg'. Four generalized mass matricies are used which have cg's between -4.0 and 24.0

```
param {
   GM1{cg(Center Of Gravity)=-4.0}
   GM2{cg=8.0}
   GM3{cg=16.0}
   GM4{cg=24.0}
   o=Mass
}
```

The next example creates a parameterized stiffness matrix for a user specified generalized coordinate. The parameter name is 'freq2' and the minimum and maximum values for the parameter are 0.0 and 10.0. The (2,2) diagonal element of mass matrix `GM2` is used to compute the frequencies for the generalized coordinate.

```
param {
   i = GSTIF001
   o = Stif
   mass=GM2
   gc=2{freq2(Second Mode Freq)[0:10]}
}
```

The output matricies from these parameterizations can be used in the stab command for parameter variation analyses.

User-written subroutine parameterizations are in demo files alge3.ax, fresp2.ax, stab6.ax, and stab12.ax. This type of parameterization is most often used to model control-laws, as in this example from stab12.ax:

```
param {
   code=usrtf
   size="stif,set=F002"
   extra=18
```

```
      o=Cont {
         gain(Control-Law Gain)[0:10] = 1
         phase(Control-Law Phase)[-60:60]<RPD> = 20
      }
   }
```

Other uses are possible as this example from stab2.ax shows:

```
   param { o=psip, size=(2,adiru), code=makepsi, datatype=real }
   alge { psi = psip{ } }
```

which creates a matrix called `psip` with 2 rows and the same number of columns as a matrix called `adiru`. The Fortran subroutine is in a data block named `makepsi` and the output matrix is forced to be real with the datatype option. The alge command creates a new matrix, `psi`, which is `psip` evaluated; because `psip` is not a function of any parameters, the empty curly braces serve to force an evaluation of `psip`, creating a matrix which is not parameterized.

ABCD control-laws are demonstrated in stab2.ax, stab7.ax, fresp2.ax and fresp4.ax. The difference between using the `ke` and `e` options is illustrated in stab2.ax:

```
   extract { ke = gstif001{col=(1,2,3)} }

   extract { E = I{col=(1,2,3), size=gstif001} }

# Create the control-law matrix with psi and ke
# and ABCD data in spline.dat...

   param {
      abcd = spline.dat {
         psi=psi
         ke=ke
         igain{igain(Input Gain)[0:5] = 1.0}
      }
      o=abcd
      plot{ [1,1] }
   }

# ...and another using the E matrix instead of ke
   param {
      abcd = spline.dat {
         psi=psi
         e = E
         igain{igain(Input Gain)[0:5] = 1.0}
      }
      o=abcdE
      plot{ [1,1] }
   }
```

10.16 print

Prints FLAPS matrices to stdout, to a specified output file, or displays the matrix graphically.

10.16.1 Syntax

print { *options* }

10.16.2 Options

See §10.1 for the general syntax of options.

matrix id { *options* }

A list of one or more matrix ids to be printed (§2.3 for information about matrix ids). A matrix id may be either a simple string if the matrix has no attributes or a string of the form

```
"name,attribute1=value1,attribute2=value2,..."
```

(in quotes to keep the commas from being interpreted as option-separators) if the matrix has attributes. The optional trailing ellipsis (three dots) means any additional attributes. The matrix id may be followed by parameter values (in curly braces) at which the matrix is to be evaluated if it has been parameterized (§10.15); for example

```
print { Mass{center=50, main=10} }
```

will print a matrix named `Mass` evaluated at two parameters (`center` and `main`).

attribute = *string*

a matrix attribute and value to be used in identifying matrices to print. If no matrix names are also included (see above option) all matrices with this attribute will be printed. For example the command

```
print { set=10 }
```

will print all matrices with `set=10`: typically `modes,set=10`, `nodes,set=10`, and `freedoms,set=10`.

o = *string*

the name of a file where the output from this command is to be written.

Default: the standard output or a graphical window depending on the size of the matrix

matview

display the matrix graphically as in figure 10.1. (only for numeric arrays)

parameters

display the current set of parameters used by all FLAPS programs.

10.16.3 Output

The rules for where and how a matrix is displayed are arcane; they depend on if the matrix is numeric (real or complex) or some other data array (nodes, freedoms, etc), the size of the matrix, and the name of the output file.

Numeric Arrays

Real and complex matrices are printed on the output file (the file specified with the o option if included, the stdout file otherwise). The format it is printed with depends on the number of columns in the matrix, the value of the pagewidth (§10.14), and the name of the output file.

If the output file name does not end in `.mm` the matrix is printed in human-readable form provided an entire row will fit within the current pagewidth. The default value of pagewidth is 130 characters, so for example a real (100,5) array would get printed in human-readable form but a (100,20) would be printed in Matrix Market (§1.1.1) format for viewing in MatView. If the pagewidth were increased to say, 500 it would be printed in human-readable format.

If an output file name is specified which ends in `.mm` the matrix will be printed to the file in Matrix Market (§1.1.1) format regardless of its size.

Non-Numeric Arrays

Non-numeric arrays such as nodal data (e.g. `nodes,set=1` or `freedoms,set=1`) or the list of defined parameters (see the `parameters` option above) are always printed on the specified output file in a human-readable format.

Graphical Display

If the `matview` option is included the matrix is displayed graphically (see figure 10.1). Alternatively if the output file ends in `.mm` the matrix will be written in Matrix Market format and it can be visualized from the command line using the matview option to the apex command (§10.6).

Clicking on an element of the matrix with the left mouse button displays the value of the element. Middle-clicking and dragging the mouse selects a region to zoom-in on, and right-clicking restores the unzoomed view.

Matrix elements are colored to indicate values according to the color map above the matrix. A slider above the color map distorts the color map; this is usually necessary to make all the non-zero elements visible.

10.16.4 Examples

The following FLAPS command from demo problem stab12.ax displays all the original generalized-aero matrices followed by the interpolated matrix evaluated at a reduced-frequency of 0.0025: in figure 10.1:

```
print { "gaf,..." }
print { pgaf{rfi=0.0025} }
```

The following FLAPS command from demo problem merge1.ax prints the nodal data and retained freedom data to stdout:

```
print{set=1, nodes, freedoms}
```

Output from this command is:

```
------------  Apex Command /home/eem2314/ax/r5/bin/print   -----------------
     print {
                 set=1, nodes, freedoms

-------------------------------------------------------


                print Version: Thu Jan 25 09:35:52 PST 2007
                      Run on Thu Jan 25 09:36:06 2007
nodes,set=1 14 by 1 Node {
    1)   1/1 (0,0,0)
    2)   2/3 (0,100,0)
    3)   3/5 (0,200,0)
    4)   4/7 (0,300,0)
    5)   5/9 (0,400,0)
    6)   6/11 (0,500,0)
    7)   7/13 (0,600,0)
    8)   101/2 (8,0,0)
    9)   102/4 (8,100,0)
   10)   103/6 (8,200,0)
   11)   104/8 (8,300,0)
```

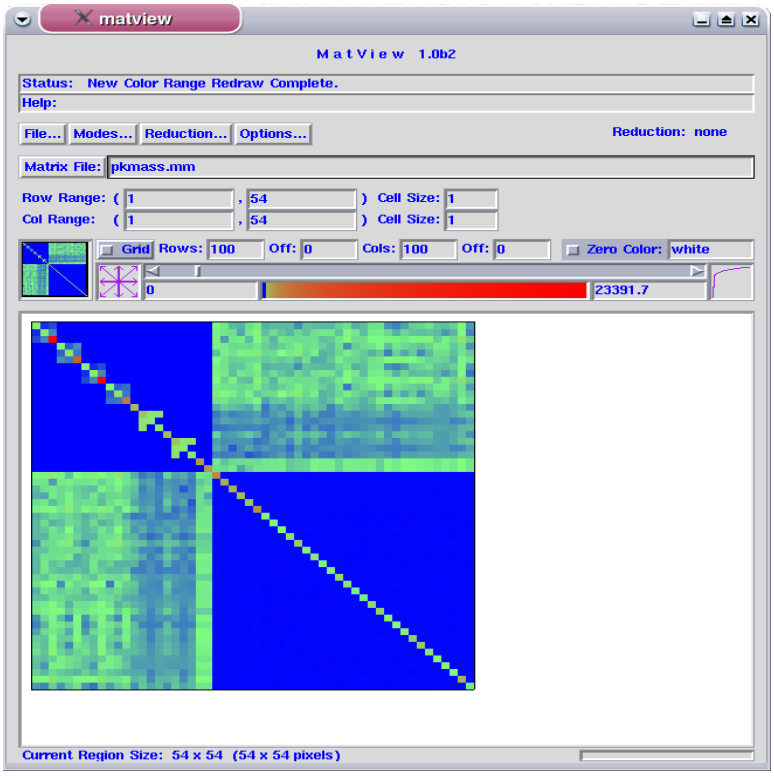


Figure 10.1: Matrix Image

```
   12)   105/10 (8,400,0)
   13)   106/12 (8,500,0)
   14)   107/14 (8,600,0)
}
freedoms,set=1 18 by 1 Freedom {
    1)   2/tz  internal node 3
    2)   2/rx  internal node 3
    3)   2/ry  internal node 3
    4)   3/tz  internal node 5
    5)   3/rx  internal node 5
    6)   3/ry  internal node 5
    7)   4/tz  internal node 7
    8)   4/rx  internal node 7
    9)   4/ry  internal node 7
   10)   5/tz  internal node 9
   11)   5/rx  internal node 9
   12)   5/ry  internal node 9
   13)   6/tz  internal node 11
   14)   6/rx  internal node 11
   15)   6/ry  internal node 11
   16)   7/tz  internal node 13
   17)   7/rx  internal node 13
   18)   7/ry  internal node 13
}
} ------------------  Apex command "print" finished: 0.01 sec
```

10.17 purge

Removes matrices from the FLAPS database

10.17.1 Syntax

purge { *matrix-name* }

10.17.2 Options

See §10.1 for the general syntax of options.

The only option is one or more matrix ids (names plus attributes); matrix ids must be in quotes (double or single) if they contain commas. Multiple matrix names must be separated by commas or newlines. Matrix ids may contain a trailing ellipsis to indicate any set of attributes. See section 2.3 for more information about matrix ids.

10.17.3 Examples

If the FLAPS database contains

```
genforce,aerocase=3,aerocond=2,rfr=0,rfi=0.02,mach=0.8
genforce,aerocase=3,aerocond=2,rfr=0,rfi=0.04,mach=0.8
genforce,aerocase=4,aerocond=2,rfr=0,rfi=0.02,mach=0.8
genforce,aerocase=4,aerocond=2,rfr=0,rfi=0.04,mach=0.8
genforce,aerocase=5,aerocond=2,rfr=0,rfi=0.02,mach=0.8
genforce,aerocase=5,aerocond=2,rfr=0,rfi=0.04,mach=0.8
genforce,aerocase=6,aerocond=2,rfr=0,rfi=0.02,mach=0.8
genforce,aerocase=6,aerocond=2,rfr=0,rfi=0.04,mach=0.8
genforce,aerocase=7,aerocond=2,rfr=0,rfi=0.02,mach=0.8
genforce,aerocase=7,aerocond=2,rfr=0,rfi=0.04,mach=0.8
```

the command

```
purge {"genforce,..."}
```

will purge all FLAPS matrices. The command

```
purge {"genforce,rfi=0.02..."}
```

will leave

```
genforce,aerocase=3,aerocond=2,rfr=0,rfi=0.04,mach=0.8
genforce,aerocase=4,aerocond=2,rfr=0,rfi=0.04,mach=0.8
genforce,aerocase=5,aerocond=2,rfr=0,rfi=0.04,mach=0.8
genforce,aerocase=6,aerocond=2,rfr=0,rfi=0.04,mach=0.8
genforce,aerocase=7,aerocond=2,rfr=0,rfi=0.04,mach=0.8
```

in the database, as will the command

```
purge {rfi=0.02}
```

10.17.4 See Also

Matrix ids in section 2.3

10.18 rename

Rename an FLAPS matrix or matrices; more generally, changes all or part of an FLAPS matrix id (§2.3). This command allows you to change the names of matrices, the names of attributes of matrices, or the values of attributes.

10.18.1 Syntax

rename { *old* to *new* }

10.18.2 Options

See §10.1 for the general syntax of options.

old

All or part of an existing matrix id that is to be replaced.

new

Replacement for all or part of the existing matrix id.

old and *new* may be the name, one or more attributes, or both. For example, to replace only the name of each matrix in a collection of unsteady aero matrices

```
rename { AEQHH to gaf }
```

will change a typical matrix id like AEQHH,rfi=0.01,rfr=0,mach=0.8 to gaf,rfi=0.01,rfr=0,mach=0.8 and likewise for all matrices with the name AEQHH.

Attributes can be renamed in the same way; for example

```
rename { rfi to kval }
```

will change the name of the attribute rfi to kval for every matrix with a matrix id containing the rfi attribute; the previous example would become AEQHH,kval=0.01,rfr=0,mach=0.8. If the value of an attribute is not specified, the existing value is used; if it is specified only matrices with that attribute/value combination will be renamed. Continuing with the previous example the command

```
rename { rfi=1e-2 to kval=0.22 }
```

will change any matrices with the attribute rfi=1e-2; the previous matrix becomes AEQHH,kval=0.22,rfr=0,mach=0.8. Note that numerical values are compared as numbers, not strings, so rfi=1e-2 is equivalent to rfi=0.01 or rfi=0.010000.

Attributes can also be removed from matrix ids by leaving them out of the new string. For example

```
rename { AEQHH,mach=0.8,rfr=0 to gf }
```

will rename all AEQHH matrices with mach=0.8, and the previous example becomes gf,rfi=0.01.

10.19 restore

Restores all FLAPS matrices previously saved using the save command.

10.19.1 Syntax

restore { *options* }

10.19.2 Options

See §10.1 for the general syntax of options.

path

file name, path or network path of a savefile previously created with the save command which contains the matrices to be restored. If the savefile is on a remote host the path is specified as with the rcp command, for example

```
restore { elgar:/acct/eem2314/ax/r5/demo/stab1.sf }
```

Default: savefile

10.19.3 Examples

the command

```
restore {$HOME/Apex/savefile}
```

restores all FLAPS matrices in subdirectory Apex/savefile below the HOME directory.

10.19.4 See Also

The save command in section 10.20

10.20 save

Writes FLAPS matrices to a savefile which is portable in the sense that it may be transferred to any other platform and read with the restore command.

10.20.1 Syntax

save { *options* }

10.20.2 Options

See §10.1 for the general syntax of options.

o = *string*

file name or path where the matrices are to be saved.

Default: savefile

matrix-id

The name of a matrix to be saved; some or all attributes may be replaced with an ellipsis. An attribute/value pair may be used to save all matrices whose name contains this attribute/value (see Examples). This option may be repeated to list all the matrices to be saved.

Default: all matrices

10.20.3 Examples

If the FLAPS database contains

```
nodes,iset=1 26 by 1 Node
freedoms,iset=1 2 by 78 int
modes,iset=1 78 by 5 Real
freqs,iset=1 5 by 5 Real
subset,iset=1,id=on1 60 by 1 int
gm1 5 by 5 Real
gm2 5 by 5 Real
gm3 5 by 5 Real
gm4 5 by 5 Real
gmass001 5 by 5 Real
gstif001 5 by 5 Real
modes001 18 by 5 Real
```

```
freqs001 5 by 5 Real
nodes,set=1 14 by 1 Node
freedoms,set=1,stage=1 2 by 18 int
coordsys,set=1 3 by 1 CoordSys
subset,set=1,id=n1 6 by 1 int
subset,set=1,id=on2 6 by 1 int
gaf,rfi=0 5 by 5 Complex
gaf,rfi=0.001 5 by 5 Complex
gaf,rfi=0.002 5 by 5 Complex
gaf,rfi=0.005 5 by 5 Complex
gaf,rfi=0.01 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=-0.01,rfi=0.001,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=0,rfi=0.001,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=0.01,rfi=0.001,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=-0.01,rfi=0.002,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=0,rfi=0.002,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=0.01,rfi=0.002,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=-0.01,rfi=0.005,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=0,rfi=0.005,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=0.01,rfi=0.005,mach=0 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=-0.01,rfi=0.001,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=0,rfi=0.001,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=0.01,rfi=0.001,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=-0.01,rfi=0.002,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=0,rfi=0.002,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=0.01,rfi=0.002,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=-0.01,rfi=0.005,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=0,rfi=0.005,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=0.01,rfi=0.005,mach=0.51 5 by 5 Complex
```

the command

```
save {o=$HOME/Apex/savefile}
```

saves all Apex matrices in subdirectory Apex/savefile below the HOME directory. The command

```
save { rfi=0.002 }
```

saves the following FLAPS matrices on a file called savefile:

```
gaf,rfi=0.002 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=-0.01,rfi=0.002,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=0,rfi=0.002,mach=0 5 by 5 Complex
genforce,aerocase=1,aerocond=2,rfr=0.01,rfi=0.002,mach=0 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=-0.01,rfi=0.002,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=0,rfi=0.002,mach=0.51 5 by 5 Complex
genforce,aerocase=2,aerocond=2,rfr=0.01,rfi=0.002,mach=0.51 5 by 5 Complex
```

and the command

```
save { "freedoms,..." }
```

will save the following on savefile:

```
freedoms,iset=1 2 by 78 int
freedoms,set=1,stage=1 2 by 18 int
```

Results from a previous stab command can be saved using the id option:

```
save {id=pk}
```

saves all results from the stab run which had id=pk. If you want to save only the targets, add to the id:

```
save { "target,id=pk" }
```

10.20.4 See Also

The restore command in section 10.19

10.21 vis

A system for viewing results from various FLAPS commands and optionally viewing animated plots of aeroelastic modes selected interactively from points on the 2-D plots. Also useful for plotting data in ESA-formatted files [15] either in an FLAPS control program or from the command line.

10.21.1 Syntax

vis { *options* }

10.21.2 Options

See §10.1 for the general syntax of options.

General Options

The source of the data to be plotted is specified using either the id option, or the esa option, or both. Either option can take multiple right-hand-sides.

id = *stringList*

Analysis id(s) from a previous execution of stab, nls, fresp, or tresp.

esa = *stringList*

ESA-formatted [34] plot file(s)

x = *stringList*

the name of a parameter (chap. 5) to be used as the independent-variable in the 2-D plot. If the word `log` is included in the list a logarithmic scale is used.

Default: fatal error

y = *stringList*

the name(s) of parameter(s) (chap. 5) to be used as the dependent-variable in the 2-D plot. If the word `log` is included in the list a logarithmic scale is used.

Default: fatal error

xmin = *float*

Minimum x-axis value. If this option is included xmax must also be included.

Default: minimum value in the x parameter data

xmax = *float*

Maximum x-axis value. If this option is included xmin must also be included.

Default: maximum value in the x parameter data

ymin = *float*

Minimum y-axis value. If this option is included ymax must also be included.

Default: minimum value in the y parameter data

ymax = *float*

Maximum y-axis value. If this option is included ymin must also be included.

Default: maximum value in the y parameter data

Animated Mode Visualization

If you want to view animated modes it is necessary to specify the source of the modal data associated with the generalized-coordinates used in the analysis given by the id option, which is also required. Modal data includes a modes matrix, a nodes matrix, a matrix of freedoms, and possibly a matrix of local coordinate systems. When a point is picked with a right-click the closest solution point is found, the generalized-coordinates at that point are multiplied by the modes matrix, then the resulting vector along with the nodes, freedoms and coordinate systems are either passed to the built-in visualizer or written to a Universal file ([25]) for subsequent visualization in amvis, ufViewer or X-Modal.

Alternatively, animated modes can be visualized in PATRAN by specifying an output file with a .op4 extension (the set, vset or iset options are not necessary). In this case a right-click creates the output file containing only the generalized-coordinates in NASTRAN OUTPUT4 format [37] which can then be imported into PATRAN for visualization.

set = *string*

nodal data set which identifies the modes matrix to be used to transform generalized-coordinates to physical dof for visualizing animated modes. If the data is from ATLAS or NASTRAN, this is an integer; if the data is from Elfini, this is the monset name.

subset = *string*

the name of a matrix consisting of a set of integer node numbers to be used in creating animated modes; only these nodes will be included in the visualization or plot file.

Default: all nodes are included

iset = *int*

Interpolation set number which was used in the ATLAS INTERP statements used to interpolate the vibration modes to the visualization grid.

vset = *int*

ATLAS vibration set number which was used in producing the data being visualized.

gc = *string*

The name of a matrix containing generalized-coordinates corresponding to the X-Y data being plotted. This option is necessary only if the data is from an ESA-formatted file, e.g. when viewing data from a response solution. It is not necessary when the fmeth or id options are included. If the X-Y data is from a response solution it is necessary to create a matrix of generalized coordinates in the fresp execution by including the option otid=gc where gc is any name you choose; then include in the vis options gc=gc

Default: generalized-coordinates are gotten from the solutions associated with the id option.

conn = *string*

Either the name of a file containing connectivity data, or the name of an ATLAS ordered nodal subset (e.g. ON90) of the visualization grid nodes. The visualization grid will be produced by drawing lines between consecutive nodes of this subset; a node number of zero means "pen-lift". If it is the name of a file, the name must be a full path name if the file is not on the current working directory or if the file is not a submit-file block.

o = *stringList*

name of a file or files where animated-mode data is to be written. The format it is written in is determined by the file extension: .op4 is OUTPUT4 format [37], .uf is Universal-file format [25], and .nf is Elfini neutral-file format.

10.21.3 Output

This command will display the requested data in a new window. Once the window is open and the data displayed the mouse buttons can be used to zoom in on portions of the data, display the values of other parameters, or to visualize aeroelastic modes:

left press the left button and sweep a region of the plot. When the button is released a new window will be created containing the outlined portion of the plot.

middle pressing the middle button will display a list of all parameter values at the point closest to the cursor; in addition, a file named .vis will be created which lists all parameter values. This file will be overwritten each time the middle mouse button is clicked.

right press the right button near the data point of interest. The closest data point will be highlighted and a new window will appear with the animated mode corresponding to that data point.

10.21.4 Discussion

The standard set of parameters which are part of all solutions in stab, fresp, or tresp are listed in table 5.1.

In addition, user-defined parameters (chap 5) may be plotted by referring to the parameter name.

Before this command can be used to visualize animated modes the user is responsible for exporting certain ATLAS, Elfini, or NASTRAN matrices

nodal data describing the visualization geometry. The SET number may be any valid ATLAS nodal data set number but the user is responsible for ensuring the SET keyword in the EXEC INTERP statements is the same as this nodal data set.

nodal subset an ordered nodal subset of the visualization grid nodes must be included in the ATLAS input data. The name of this subset is then passed to vis using the grid keyword.

interp EXEC INTERP statements must be included in the ATLAS control program to interpolate from the vibration modes to the visualization grid.

10.21.5 Command Line Usage

Both vis and amvis can be used on the command line to visualize data in ESA-formatted or Universal files, respectively. To avoid having to know the location of these commands it is convenient to run them from the apex script, e.g.

```
$ apex vis ...
$ apex amvis ...
```

The options to vis are similar to the control-program version; the main difference is that the syntax is like other Unix commands. For example, to plot growth against veas from a file called pk.esa:

```
$ apex vis -x veas -y growth pk.esa
```

Not all options are available in the command-line version; animated-mode visualization cannot be driven from the command-line version because the geometry data is not in ESA files, so set, subset, iset, vset, gc, o, and conn are ignored.

Multiple file may be specified; the number of files is only limited by computer memory and the number of arguments allowed on a command-line by the operating system (hundreds).

Here is a synopsis of the command-line version of vis:

```
apex vis -x xpar -y ypar [options] file(s)
    -x xpar     xpar is the name of the x-variable
    -y list     the name(s) of the y-variable(s).  Multiple runs can be specified
                using an Apex to-by list enclosed in quotes. For example
                -y '(x1 to x12 by 2)'
    -xmin n     n is the minimum x-value
    -xmax n     n is the maximum x-value
    -ymin n     n is the minimum y-value
    -ymax n     n is the maximum y-value
    -color      give all curves from a file the same color. Useful
                when plotting multiple files.
    -x log      plot the x-axis on a log scale
    -y log      plot the y-axis on a log scale
    -r list     runid(s) to plot. Multiple runs can be specified
                using an Apex to-by list enclosed in quotes. For example
                -r '(mode_3 to mode_12 by 2)'
```

There is only one option to amvis: the name (or more precisely the path) of the Universal file. If the name of the file ends in `.uf` it is not necessary to include the extension, so

```
apex amvis myfile.uf
```

and

```
apex amvis myfile
```

are equivalent.

10.21.6 Examples

The command

```
vis { id=pk, x=vtas, y=growth, set=1 }
```

in an FLAPS control program will produce a plot from a previous stab run; clicking on the curves will produce animated modes base on `set 1` data from BAP or ATLAS.

To plot on a log scale, include the word `log` in the list of `x` or `y` parameters:

```
vis { id=frf, x=(freq,log), y=(pilot,log) }
```

10.21.7 See Also

- FLAPS command stab (10.10)
- FLAPS command fresp (**??**)
- FLAPS command tresp (**??**)

Part III

Appendices

Appendix A

Creating Flaps Savefiles

This appendix contains instructions for creating Flaps savefiles from Elfini or ATLAS; Flaps savefiles can be created from BAP databases with the import option to the apex command (§10.12.3) or BAP data can be imported directly into an Flaps control program using the import command (§10.12). Savefiles created using any of these techniques are portable in the sense that they can be transferred to any platform and used in Flaps.

Binary files created on one type of computer are generally not usable on different types of computer. Computers are classified according to their *architecture* or the type of CPU. For example, an IBM RS/6000 uses a PowerPC CPU, and the Linux clusters use AMD CPUs. Both these architectures use a format for storing floating-point numbers which adhere to the IEEE standard [6], but binary files created on one are not usable on the other because the bytes which make up a floating-point number are ordered differently on these two architectures [48].

The External Data Representation Standard [46] [49] (XDR) was created to allow programs to transmit data across the network between different platforms. This standard consists of rules for encoding various types of data (floating point, double precision, integer, etc) and for decoding data. Flaps uses XDR for transmitting data between programs, for storing data in its database, and in its savefiles. This is why Flaps savefiles are portable across architectures and why Flaps commands running on different architectures can communicate with each other. Another consequence of this design is that Flaps databases consist of a directory containing all the data, each array in a separate file, and each of these files is an Flaps savefile containing only one array.

A.1 Elfini

An Flaps savefile can be created from an existing Elfini database with the extract option to the apex command. Input to this command is from stdin, which in a script is conveniently provided using a "here document" (see chapter 2). Input to this command is a set of options similar to other Flaps commands. The options are

f30 = string path of the Elfini database

model = string Elfini model name

monset = string monset to include in the savefile; node and freedom information are also included.

stif include the generalized stiffness matrix associated with each specified monset

mass = string include the generalized mass matrix associated with this mass case

A.1.1 Example

```
apex -v alpha extract <<@
        f30 = /sd/catel/elf_file
        model=B002
        stif
        monset=(AS01,CLAW)


#  BASIC OEW AND PAYLOAD MASS CASES
#  --------------------------------
          mass=(PLO1 # 777 FREIGHTER WITH NOMINAL BODY DESIGN OEW AND NO PAYLOAD
                PLO3 # 777 FREIGHTER LIGHT-WEIGHT BODY OEW AND NO PAYLOAD
                PLA1 # A1 End Loaded Forward C.G. Payload
                PLB1 # B1 Center Loaded Forward C.G. Payload
                PLC1 # C1 End Loaded Aft C.G. Payload
                PLP1 # P1 Fatigue Payload

#  CENTER FUEL CASES
#  -----------------
                CO20 # 20% CENTER TANK FUEL (7.1 lb/gal DENSITY)
                CO40 # 40% CENTER TANK FUEL (7.1 lb/gal DENSITY)
                CO60 # 60% CENTER TANK FUEL (7.1 lb/gal DENSITY)
                CO80 # 80% CENTER TANK FUEL (7.1 lb/gal DENSITY)
                C100 #100% CENTER TANK FUEL (7.1 lb/gal DENSITY)

#  MAIN (WING) FUEL CASES
#  ----------------------
                M005 #  5% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M010 # 10% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M015 # 15% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M020 # 20% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M025 # 25% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M030 # 30% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M035 # 35% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M040 # 40% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M045 # 45% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M050 # 50% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M055 # 55% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M060 # 60% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M065 # 65% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M070 # 70% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M075 # 75% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M080 # 80% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M085 # 85% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M090 # 90% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M095 # 95% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)
                M100 #100% MAIN (WING) TANK FUEL (7.1 lb/gal DENSITY)

#  CONTROL SURFACE PAINT & REPAIR CASES
#  ------------------------------------
```

```
                PXLA # LEFT AILERON REPAINT (OSML)
                PXRA # RIGHT AILERON REPAINT (OSML)
                LAAW # LEFT AILERON ADJUST WEIGHTS
                RAAW # RIGHT AILERON ADJUST WEIGHTS
                LATW # LEFT AILERON TIP REPAIR POINT MASS AT TE
                RATW # RIGHT AILERON TIP REPAIR POINT MASS AT TE
                LARW # LEFT AILERON ROOT REPAIR POINT MASS AT TE
                RARW # RIGHT AILERON ROOT REPAIR POINT MASS AT TE
                PALA # LEFT AILERON DISTR. PAINT AND REPAIR + ADJUST WEIGHTS (OSML)
                PARA # RIGHT AILERON DISTR. PAINT AND REPAIR + ADJUST WEIGHTS (OSML)


                PALE # LEFT ELEVATOR DISTR. PAINT AND REPAIR (OSML)
                PARE # RIGHT ELEVATOR DISTR. PAINT AND REPAIR (OSML)
                PTLE # LEFT ELEVATOR DISTR. PAINT AND REPAIR + TIP REPAIR (OSML)
                PTRE # RIGHT ELEVATOR DISTR. PAINT AND REPAIR + TIP REPAIR (OSML)

                PARX # RUDDER DISTRIBUTED PAINT AND REPAIR (OSML)
                PARY # RUDDER DISTRIBUTED PAINT AND REPAIR + TIP REPAIR (OSML)
                PART # TAB DISTRIBUTED PAINT AND REPAIR (OSML)

                F10L # 10% LEFT FLAPERON MASS
                F10R # 10% RIGHT FLAPERON MASS

#   777 FREIGHTER WITH LIGHT-WEIGHT BODY OEW AND 80% MAIN TANK FUEL
#  ------------------------------------------------------------
                DICK

#  DUMMY CASE WITH MASS ON FRONTIER NODES
#  --------------------------------------
                FDUM

#   ICE CASES
#   ---------
                WICE # WING ICE
                HICE # HORIZ. STAB ICE
                FICE # FIN ICE


  )

@
```

A.2 ATLAS

To create a savefile from ATLAS data include an ATLAS CALL SAVE statement in an ATLAS control program. In order to use this command it is necessary to use a special version of the ATLAS script on /acct/eem2314/bin. Demo problem stab123.q is an ATLAS job that creates an FLAPS savefile from a simple ATLAS beam model.

The following documentation is from the source file for the ATLAS SAVE subroutine; the current version is kept in $AXROOT/src/lib/libatlas.

```
 ROUTINE:     SAVE

 AUTHOR:      Ed Meyer

 DATE:        March 2005

 PURPOSE:     To write one or more atlas matrices to a
```

```
            Apex binary file which may be transferred
            between computers.

USAGE:      ATLAS CALL SAVE (PLIST)
            The plist consists of the following order-
            independent keywords. only the first four
            characters need be included.

PARAMETERS: I N P U T
            ---------


        GAF=(A,B,C) - Aero matrix or interpolation coefficient
                      identifier.  A is the name of an aerodynamic
                      module (e.g. AF1 or DUBLAT), B is the aero
                      case number, and C is the aero condition number.
                      If B and C are not included, A is assumed to be
                      the name of a set of ADDINT interpolation
                      coefficients. In this case only the original
                      gaf matrices are saved.
                      Default: no aero matrices are included

        ISET=N      - ATLAS interpolation set number. If this keyword
                      is included, the ISET modes, nodal data, and
                      freedom data will be included, which are
                      necessary for visualizing flutter modes.
                      Default: no interpolation data is included.

        NAME        - The name of an ATLAS user matrix.
                      If the matrix has attributes
                      the attributes/value pairs must
                      follow immediately after the name.
                      For example, to read the KRFV for
                      SET 3, STAGE 5, include
                      ATLAS CALL SAVE (..,KRFV,SET=3,STAGE=5,..)

        O=file      - The name of the file to receive data
                      Default: o=savefile

        SET=N       - ATLAS nodal data set number. If this keyword
                      is included, nodal data matrices will be
                      included, which are necessary for running
                      Apex dublat, or for visualizing flutter modes.
                      Default: no nodal data is included.

        SS=N        - Substructure number. As an alternative to
                      specifying SET/STAGE, if SS is included, nodal
                      data for this substructure will be included.
                      Default: SS=0

        STAGE=N     - BC stage number.
                      Default: STAGE=1 if SET keyword is included,
                      STAGE=0 otherwise.

        VSET=N      - Vibration set number. If vset is included,
                      GMASS, GSTIF, FREQS and MODES are saved.
                      Default: VSET matrices are not saved


            O U T P U T
            -----------

            Output consists of a binary file with the name given by the
            'O' option. This file is "portable": it may be copied to other
            platforms and imported into Apex.
```

```
FILES:       binary file with the name given by
             the 'O' plist keyword. This file is in XDR format
             so it is portable across platforms.

SUBROUTINES:  GETSET, GFCIN, NODAT, RDSPUM, CHECKP, GETIND (ALIB)
              AVLUN, MATIND, UMINF (ALIB)
              DOEXP (CLIB)

LANGUAGE:    ATLAS CONTROL SUBROUTINE

EXAMPLE:     The ATLAS control program statement
                ATLAS CALL SAVE (VSET=10,FMAT,GMAT,
               *       O=JUNK,GAF=(DUBL,1,2))
             will create a file named 'JUNK' containing
             GMASS010, GSTIF010, FREQS010, MODES010, user matrices
             GMAT and FMAT, and generalized aero matrices from DUBLAT
             CASE=1, COND=2. These aero matrices will have names like
                GENFORCE,AEROCASE=1,AEROCOND=2,MACH=0,PVALR=0,PVALI=0.063
             They are typically passed to the Apex param processor to create
             interpolation coefficients for use in the stab flutter solver:
                param { genforce, o=gaf }
```

Appendix B

User-Written Subroutines

This appendix documents subroutines supplied by FLAPS which are callable from user-written Fortran subroutines.

B.1 Fortran Subroutines

B.1.1 fetch

Fetch a matrix from the FLAPS data manager.

Syntax

```
character*8 dtype
integer nr, nc, offset, irr
real a(1)
call fetch ('name', nr, nc, a, offset, dtype, iprint, irr)
if (irr .ne. 0) then
   call feterr
     ...
endif
```

Input

name (character string) matrix name

a a real array dimensioned at least 1.

iprint debug option; if iprint is greater than zero the matrix will be printed.

Output

nr number of rows in the matrix

nc number of columns in the matrix

offset the index in array a of the start of the matrix. If the matrix is real the (1,1) element is in a(offset), followed by the (2,1) element in a(offset+1) (standard Fortran storage). If the matrix is complex the real part of the (1,1) element is in a(offset) followed by the imaginary part of the (1,1) element in a(offset+1), followed by the real part of the (2,1) element in a(offset+2), and the imaginary part of the (2,1) element in a(offset+3), etc.

dtype (character*1) datatype of the output matrix: 'R' for real matrix, 'C' for complex.

irr error flag; if irr is not zero the matrix could not be read.

Discussion

Because Fortran does not have dynamic memory allocation, and to avoid having to pass a fixed-size array to fetch, all that is passed is the name of a real or complex array of length one. Memory is allocated in fetch, which is written in C, then an offset relative to the input array is returned, allowing the actual memory to be accessed using the input array. For example, if you know the matrix is real, you can write

```
real a(1)
integer offset
   ...
call fetch ('sensor',nr,nc,a,offset,dtype,iprint,irr)
```

then you can access the first element of the array with, for example

```
a11 = a(offset)
```

and the (i,j) element with, for example

```
ij = i + nr*(j-1)
aij = a(offset-1+ij)
```

where ij reflects the way memory is layed out in Fortran (by columns).

If you are uncertain of the datatype of the matrix you can force a real and a complex array to have the same memory location with an equivalence statement as in

```
real ra(1)
complex ca(1)
equivalence (ra(1),ca(1))
call fetch ('sensor', nr, nc, ra, offset, dtype, iprint, irr)
```

then you can access the first element of the array with, for example

```
      if (dtype(1:1) .eq. 'r') then
         a11 = ra(offset)
      else
         a11 = ca(offset)
      endif
```

Subroutine feterr can be called to print a message giving more details about why the fetch was not successful. It takes no arguments and only prints (on stdout) a message then returns.

B.1.2 parval

The current value in internal units of any user-defined (§5.6) or standard parameter (§5.1) can be retrieved with the parval subroutine.

Syntax

```
real value
value = parval('name')
```

Example

```
complex s
s = cmplx(parval('sigma'), parval('freq'))
```

sets the complex characteristic exponent s to it's current value.

B.1.3 getpar

gets the value (in external units) of a user-defined or standard parameter.

Syntax

```
real value
value = getpar('name')
```

Example

```
real value
value = getpar('vtas')
```

gets the current value of vtas in knots.

B.1.4 setpar

sets the value (in external units) of a user-defined or standard parameter.

Synopsis

```
real value
call setpar(value, 'name')
```

Example

```
real value
value = 400.0
call setpar(value, 'vtas')
```

sets the value of vtas to 400 knots.

B.1.5 denom2, denom3, denom4

These functions are intended to be used to find poles of controls equations consisting of rational polynomials in the characteristic exponent s. A rational polynomial is a polynomial (the numerator) divided by another polynomial (the denominator). If s approaches a root of the denominator polynomial, the denominator approaches zero and the rational polynomial approaches infinity (known as a *pole* of the rational polynomial), leading to severe numerical problems. While the chances of an aeroelastic root coming close to a denominator root are slim, it is not impossible as demo problem stab12.ax shows. If the denominator polynomials are evaluated using one of these functions an ESA-formatted plot file named `poles.esa` will be created containing the zeros of each polynomial evaluated using these functions. If the aeroelastic modes are plotted freq against sigma together with poles.esa, it becomes immediately apparent if one of the poles is causing problems.

Syntax

```
complex denom2, denom3, denom4
complex s, t
real a1, a2, a3, a4
t = denom2(a1,a2,s)
t = denom3(a1,a2,a3,s)
t = denom4(a1,a2,a3,a4,s)
```

Input

a1, a2, a3, a4 (real) coefficients of the denominator polynomial

Example

```
complex s
s = cmplx(parval('sigma'),parval('freq'))
complex denom2, denom3, denom4
tf2 = 1/denom2(2.0, 3.0, s)
tf3 = 1/denom3(2.0, 3.0, 4.0, s)
tf4 = 1/denom4(2.0, 3.0, 4.0, 5.0, s)
```

is equivalent to

```
complex s
s = cmplx(parval('sigma'),parval('freq'))
tf2 = 1/(2.0*s + 3.0)
tf3 = 1/(2.0*s*s + 3.0*s + 4.0)
tf4 = 1/(2.0*s*s*s + 3.0*s*s + 4.0*s + 5.0)
```

Appendix C

ABCD Approach to ASE Analysis

C.1 Controls Equations

The traditional form for controls equations is the set of state-space equations

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{C.1}$$

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \tag{C.2}$$

where $\boldsymbol{x}$ is an n_s -vector of states, $\boldsymbol{u}$ is an n_i -vector of inputs, $\boldsymbol{y}$ is an n_o -vector of outputs, and the matrices have dimensions:

$$\begin{aligned} \boldsymbol{A} &: (n_s, n_s) \\ \boldsymbol{B} &: (n_s, n_i) \\ \boldsymbol{C} &: (n_o, n_s) \\ \boldsymbol{D} &: (n_o, n_i) \end{aligned}$$

If there are input and output time delays the controls equations can be written

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{G}\boldsymbol{u} \tag{C.3}$$

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{C}\boldsymbol{x} + \boldsymbol{H}\boldsymbol{D}\boldsymbol{G}\boldsymbol{u} \tag{C.4}$$

where

$$\boldsymbol{G} = \begin{bmatrix} \ddots & & \\ & g_i e^{-st_i} & \\ & & \ddots \end{bmatrix} \tag{C.5}$$

and

$$\boldsymbol{H} = \begin{bmatrix} \ddots & & \\ & h_j e^{-st_j} & \\ & & \ddots \end{bmatrix} \tag{C.6}$$

are diagonal matrices, g_i and t_i are the (complex) gain and time delay associated with the i^{th} input channel, and h_j and t_j are the (complex) gain and time delay associated with the j^{th} output channel.

In addition to input and output time delays there may be *internal* time delays which are represented by exponentials multiplying terms of the $\boldsymbol{A}$ matrix; each term may be multiplied by multiple time delays and each time delay may multiply multiple elements:

$$\begin{aligned} \boldsymbol{A}(s) &= \left[A_{ij} e^{-st_{ij}}\right] \\ t_{ij} &= \sum_{l=1}^{m_{ij}} t_{k_l} \end{aligned} \tag{C.7}$$

Moreover, the $\boldsymbol{A}$ matrix may be a function of altitude or true airspeed so in general $\boldsymbol{A}$ is written $\boldsymbol{A}(s, z)$ or $\boldsymbol{A}(s, V_t)$.

C.2 Structural Equations

The characteristic equations for a discrete n_e degree of freedom linear structure including unsteady aerodynamic terms are usually written as

$$\left[s^2\boldsymbol{M} + s\boldsymbol{V} + \boldsymbol{K} - q\boldsymbol{Q}\right]\boldsymbol{q} = \boldsymbol{f} \tag{C.8}$$

where $\boldsymbol{q}$ is a (complex) n_e -vector of generalized coordinates, s is the complex characteristic exponent, $\boldsymbol{M}, \boldsymbol{V}, \boldsymbol{K}$, and $\boldsymbol{Q}$ are the (n_e, n_e) generalized mass, damping, stiffness, and unsteady aerodynamic matrices, respectively, q is the dynamic pressure, and $\boldsymbol{f}$ is a set of external forces.

C.3 Combining Structural and Controls Equations

In order to combine the control-law equations with the structural equations it is first necessary to relate the input ($\boldsymbol{u}$) and output ($\boldsymbol{y}$) of the controls equations to structural degrees of freedom.

The control-law input $\boldsymbol{u}$ is related to the generalized-coordinates $\boldsymbol{q}$ by

$$\boldsymbol{u} = \boldsymbol{S}\Psi\boldsymbol{q} \tag{C.9}$$

where Ψ is an (n_i, n_e) transformation matrix, usually consisting of rows of the modes matrix (the basis for the generalized coordinates), and $\boldsymbol{S}$ is an (n_i, n_i) diagonal matrix whose elements are either 1, s or s^2 depending on whether the corresponding element of $\boldsymbol{u}$ is a displacement, velocity or acceleration.

Substituting C.9 into C.1

$$\boldsymbol{BS}\Psi\boldsymbol{q} + (\boldsymbol{A} - s\boldsymbol{I})\,\boldsymbol{x} = 0 \tag{C.10}$$

If the control-law output $\boldsymbol{y}$ is related kinematically to the generalized coordinates by an (n_e, n_o) matrix $\boldsymbol{E}$ then the external force applied to the structure due to $\boldsymbol{y}$ is

$$\boldsymbol{f} = \boldsymbol{KEy} \tag{C.11}$$

Substituting equations C.2 and C.9

$$\boldsymbol{f} = \boldsymbol{KE}\,(\boldsymbol{Cx} + \boldsymbol{Du}) = \boldsymbol{KE}\,(\boldsymbol{Cx} + \boldsymbol{DS}\Psi\boldsymbol{q}) \tag{C.12}$$

Combining C.8, C.10 and C.12

$$\begin{bmatrix} s^2\boldsymbol{M} + s\boldsymbol{V} + \boldsymbol{K} - q\boldsymbol{Q} - \boldsymbol{KEDS}\Psi & -\boldsymbol{KEC} \\ \boldsymbol{BS}\Psi & \boldsymbol{A} - s\boldsymbol{I} \end{bmatrix} \begin{Bmatrix} \boldsymbol{q} \\ \boldsymbol{x} \end{Bmatrix} = 0 \tag{C.13}$$

or, if input and output gains and time-delays are included

$$\begin{bmatrix} s^2\boldsymbol{M} + s\boldsymbol{V} + \boldsymbol{K} - q\boldsymbol{Q} - \boldsymbol{KEHDGS}\Psi & -\boldsymbol{KEHC} \\ \boldsymbol{BGS}\Psi & \boldsymbol{A} - s\boldsymbol{I} \end{bmatrix} \begin{Bmatrix} \boldsymbol{q} \\ \boldsymbol{x} \end{Bmatrix} = 0 \tag{C.14}$$

When the input gain ($\boldsymbol{G}$) is zero, the second row of equation C.14 reduces to the eigenvalue problem

$$\boldsymbol{Ax} - s\boldsymbol{I} = \boldsymbol{0} \tag{C.15}$$

and x is zero except when s is an eigenvalue of $\boldsymbol{A}$; everywhere else the equations reduce to the open-loop structural equations, as they should.

A more useful form is

$$\left[s^2\bar{\boldsymbol{M}} + s\bar{\boldsymbol{V}} + \bar{\boldsymbol{K}} - q\bar{\boldsymbol{Q}} + \bar{\boldsymbol{T}}\right]\bar{\boldsymbol{q}} = 0 \tag{C.16}$$

where

$$\bar{\boldsymbol{M}} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},\ \bar{V} = \begin{bmatrix} \boldsymbol{V} & \boldsymbol{0} \\ 0 & 0 \end{bmatrix},\ \bar{\boldsymbol{K}} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},\ \bar{\boldsymbol{Q}} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},\ \bar{\boldsymbol{q}} = \begin{Bmatrix} \boldsymbol{q} \\ \boldsymbol{x} \end{Bmatrix} \tag{C.17}$$

and

$$\bar{\boldsymbol{T}} = \begin{bmatrix} -\boldsymbol{KEHDGS}\Psi & -\boldsymbol{KEHC} \\ \boldsymbol{BGS}\Psi & \boldsymbol{A} - sI \end{bmatrix} \tag{C.18}$$

Equation C.16 is the form of the flutter equations used in Apex.

Appendix D

ABCD File Format

ABCD files are usually produced in Matlab/SIMULINK by the flight controls group. They are ASCII files so you can edit them and move them across platforms; if you edit them on a Windows machine you may have to run a program called dos2unix prior to using it in FLAPS.

Comment lines start with a number sign (#) and are ignored. The format is free-field: numbers do not need to appear in specific columns of a line.

In the following description of the file format a number of variables are used:

ns the number of rows and columns in the A matrix, the number of rows in B and the number of columns in C

ni the number of inputs; the number of columns in B and D.

no the number of outputs; the number of rows in C and D.

nb the number of breakpoints in the A matrix interpolant.

nix the number of elements of A which are interpolated

nin the number of elements of A which are constant

ntd the number of internal time delays

The file consists of blocks of data:

1 line 6 integers: ns ni no nb nix nin

1 line (string) Name of the parameter used to interpolate the A matrix (e.g. alt or vcas)

nb lines (1 float per line) values of the variable used to interpolate A at the breakpoints

nix lines (2 integers per line) row and column indices of the elements of A which are interpolated

nin lines (2 integers per line) row and column indices of the elements of A which are constant (seems redundant).

ni lines (1 float per line) input time delays

no lines (1 float per line) output time delays

ni lines (1 float per line) exponents of s in the S matrix (see C.9).

ns*ns lines (1 float per line) elements of the A matrix by rows

ns*ni lines (1 float per line) elements of the (ns,ni) B matrix by rows

no*ns lines (1 float per line) elements of the (no,ns) C matrix by rows

no*ni lines (1 float per line) elements of the (no,ni) D matrix by rows

(nb-1)*4*nix lines (1 float per line) coefficients for interpolating the A matrix: a (nb-1, 4, nix) array stored as

```
(((coef(i,j,k), j=1,4), i=1,nb-1), k=1,nix)
```

1 line (1 integer) number of internal time delays (ntd). Following this line will be ntd repetitions of the next 3 blocks

1 line (1 integer) number of elements of A affected by this time delay (ntdi)

ntdi lines (2 integers per line) row and column indices of the elements of A which are affected by this time delay.

ntd lines (1 float per line) time delays

Appendix E

Rational-Function Approximation

The param command can approximate generalized unsteady aerodynamic matrices using a rational-function approximation often referred to as *s-plane* aerodynamics, even though the approximation is done with respect to complex reduced-frequency p, not s. This approximation, due to Roger [42], uses a least-squares fit of a certain rational-polynomial to a set of complex matrices. Given a set of m unsteady aero matrices at various values of complex reduced frequency p

$$\boldsymbol{Q}_k = \boldsymbol{Q}(p_k) \quad k = 1, m \tag{E.1}$$

where

$p = \frac{s}{v_t}$ is the complex reduced-frequency, s is the Laplace variable, and v_t is the true airspeed, approximate $\boldsymbol{Q}$ with an analytic function of p. Roger's approximation to the aero matrix has the form

$$\boldsymbol{Q}(p) \approx \boldsymbol{R}_0 + p\boldsymbol{R}_1 + p^2\boldsymbol{R}_2 + \sum_{i=1}^{\ell} \frac{p}{p+\beta_i}\boldsymbol{R}_{i+2} \tag{E.2}$$

where the matrices $\boldsymbol{R}_i$ are real, and the β_i are real.

To determine the matrices $\boldsymbol{R}_i$, write m equations, one for each aero matrix $\boldsymbol{Q}_k$:

$$\begin{bmatrix} 1 & p_1 & p_1^2 & \frac{p_1}{p_1+\beta_1} & \cdots & \frac{p_1}{p_1+\beta_\ell} \\ 1 & p_2 & p_2^2 & \frac{p_2}{p_2+\beta_1} & \cdots & \frac{p_2}{p_2+\beta_\ell} \\ & & & \ddots & & \\ 1 & p_m & p_m^2 & \frac{p_m}{p_m+\beta_1} & \cdots & \frac{p_m}{p_m+\beta_\ell} \end{bmatrix} \begin{bmatrix} r_{0,11} & r_{0,21} & \cdots & r_{0,nn} \\ r_{1,11} & r_{1,21} & \cdots & r_{1,nn} \\ & & \ddots & \\ r_{\ell+3,11} & r_{\ell+3,21} & \cdots & r_{\ell+3,nn} \end{bmatrix}$$

$$= \begin{bmatrix} q_{0,11} & q_{0,21} & \cdots & q_{0,nn} \\ q_{1,11} & q_{1,21} & \cdots & q_{1,nn} \\ & & \ddots & \\ q_{m,11} & q_{m,21} & \cdots & q_{m,nn} \end{bmatrix} \tag{E.3}$$

where $r_{k,ij}$ is the (i,j) element of the real (n,n) matrix $\boldsymbol{R}_k$ and $q_{k,ij}$ is the (i,j) element of the complex (n,n) matrix $\boldsymbol{Q}_k$.

Expanding the complex terms into real and imaginary,

$$\begin{bmatrix} 1 & \Re(p_1) & \Re(p_1^2) & \Re(\frac{p_1}{p_1+\beta_1}) & \dots & \Re(\frac{p_1}{p_1+\beta_\ell}) \\ 0 & \Im(p_1) & \Im(p_1^2) & \Im(\frac{p_1}{p_1+\beta_1}) & \dots & \Im(\frac{p_1}{p_1+\beta_\ell}) \\ 1 & \Re(p_2) & \Re(p_2^2) & \Re(\frac{p_2}{p_2+\beta_1}) & \dots & \Re(\frac{p_2}{p_2+\beta_\ell}) \\ 0 & \Im(p_2) & \Im(p_2^2) & \Im(\frac{p_2}{p_2+\beta_1}) & \dots & \Im(\frac{p_2}{p_2+\beta_\ell}) \\ & & & & \ddots & \\ 1 & \Re(p_m) & \Re(p_m^2) & \Re(\frac{p_m}{p_m+\beta_1}) & \dots & \Re(\frac{p_m}{p_m+\beta_\ell}) \\ 0 & \Im(p_m) & \Im(p_m^2) & \Im(\frac{p_m}{p_m+\beta_1}) & \dots & \Im(\frac{p_m}{p_m+\beta_\ell}) \end{bmatrix} \begin{bmatrix} r_{0,11} & r_{0,21} & \cdots & r_{0,nn} \\ r_{1,11} & r_{1,21} & \cdots & r_{1,nn} \\ & & \ddots & \\ r_{\ell+3,11} & r_{\ell+3,21} & \cdots & r_{\ell+3,nn} \end{bmatrix}$$

$$= \begin{bmatrix} \Re(q_{0,11}) & \Re(q_{0,21}) & \dots & \Re(q_{0,nn}) \\ \Im(q_{0,11}) & \Im(q_{0,21}) & \dots & \Im(q_{0,nn}) \\ \Re(q_{1,11}) & \Re(q_{1,21}) & \dots & \Re(q_{1,nn}) \\ \Im(q_{1,11}) & \Im(q_{1,21}) & \dots & \Im(q_{1,nn}) \\ & & \ddots & \\ \Re(q_{m,11}) & \Re(q_{m,21}) & \dots & \Re(q_{m,nn}) \\ \Im(q_{m,11}) & \Im(q_{m,21}) & \dots & \Im(q_{m,nn}) \end{bmatrix} \tag{E.4}$$

Which is a set of real linear equations

$$\boldsymbol{AX} = \boldsymbol{B} \tag{E.5}$$

where $\boldsymbol{A}$ is $(2m, 3+\ell)$, $\boldsymbol{X}$ is $(3+\ell, n^2)$, and $\boldsymbol{B}$ is $(2m, n^2)$. These equations are overdetermined if $2m > 3+\ell$, that is if the number of matrices $\boldsymbol{Q}_k$ is greater than the number of β_i plus three divided by 2.

Appendix F

Interpolation Details

This appendix contains some details about the interpolation techniques used in FLAPS.

F.1 Smoothing TPS Limits

Tensor Product Splines (TPS) are splines in one or more dimensions; for example if unsteady aerodynamic matrices are available at multiple reduced frequencies and multiple Mach numbers they can be interpolated with respect to both reduced-frequency and Mach number. There is no limit (except perhaps memory) on the number of dimensions possible in a TPS. In the simplest (and most common) case there is only one dimension in the TPS and the resulting TPS is identical to an ordinary cubic spline.

F.1.1 Smoothing TPS Limits

At the boundaries of tensor-product spline (TPS) interpolations derivatives are set to zero and the value is held constant outside the range of interpolation to avoid the unrealistically large values common with extrapolation. Unfortunately, unlike extrapolation holding the derivative constant outside the limits of interpolation causes a discontinuity in the derivative, which can cause problems in stab which depends on up to first derivative continuity, so to remove this discontinuity the transition from in-range values to out-of-range values is smoothed.

If you interpolate a set of matrices $(\mathbf{A}_1, \mathbf{A}_2, ...\mathbf{A}_n)$ at a set of breakpoints $x_i, i = 1, ...n$ then we want to fit a cubic polynomial in the interval $[x_1 - w, x_1]$ and another between $[x_n, x_n + w]$. Outside these intervals ($x < x_1 - w$ or $x > x_n + wx$) the spline is considered constant. For a distance w outside the upper limit of the interpolation let $t = x - x_n$ and assume an equation for the polynomial:

$$\mathbf{G}(t) = t^3\mathbf{G}_3 + t^2\mathbf{G}_2 + t\mathbf{G}_1 + \mathbf{G}_0 \tag{F.1}$$

The coefficient matrices $\boldsymbol{G}_i$ can be solved for by enforcing the conditions that the function and it's first derivative are continuous at $t = 0$, the value at $t = w$ is a small

extrapolation of the spline,

$$\mathbf{G}(w) = \mathbf{A}_n + \beta \mathbf{A}'_n \tag{F.2}$$

and the first derivative is zero at $t = w$. These conditions result in

$$\begin{aligned}
\mathbf{G}(0) &= \mathbf{G}_0 = \mathbf{A}(x_n) \\
\mathbf{G}'(0) &= \mathbf{G}_1 = \mathbf{A}'(x_n) \\
\mathbf{G}(w) &= \mathbf{A}_n + \beta \mathbf{A}'_n \\
&= w^3\mathbf{G}_3 + w^2\mathbf{G}_2 + w\mathbf{G}_1 + \mathbf{G}_0 \\
&= w^3\mathbf{G}_3 + w^2\mathbf{G}_2 + w\mathbf{A}'_n + \mathbf{A}_n \\
\mathbf{G}'(w) &= \mathbf{0} \\
&= 3w^2\mathbf{G}_3 + 2w\mathbf{G}_2 + \mathbf{G}_1 \\
&= 3w^2\mathbf{G}_3 + 2w\mathbf{G}_2 + \mathbf{A}'_n
\end{aligned} \tag{F.3}$$

From which

$$\begin{aligned}
\mathbf{G}_2 &= \left(\frac{3\beta}{w^2} - \frac{2}{w}\right)\mathbf{A}'_n \\
\mathbf{G}_3 &= \frac{1}{3w^3}\left(3w - 6\beta\right)\mathbf{A}'_n
\end{aligned} \tag{F.4}$$

and

$$\begin{aligned}
\mathbf{G}(t) &= \left[t^3\left(\frac{1}{3w^3}\left(3w - 6\beta\right)\right) + t^2\left(\frac{3\beta}{w^2} - \frac{2}{w}\right) + t\right]\mathbf{A}'_n + \mathbf{A}_n \\
\mathbf{G}'(t) &= \left[t^2\left(\frac{1}{w^3}\left(3w - 6\beta\right)\right) + 2t\left(\frac{3\beta}{w^2} - \frac{2}{w}\right) + 1\right]\mathbf{A}'_n
\end{aligned} \tag{F.5}$$

If $\beta = w$ equation F.5 simplifies to

$$\begin{aligned}
\boldsymbol{G}(t) &= \left(-\frac{t^3}{w^2} + \frac{t^2}{w} + t\right)\boldsymbol{A}'_n + \boldsymbol{A}_n \\
\boldsymbol{G}'(t) &= \left(-\frac{3t^2}{w^2} + \frac{2t}{w} + 1\right)\boldsymbol{A}'_n
\end{aligned} \tag{F.6}$$

Figures F.1 and F.2 illustrate this smoothing polynomial at the upper limit of a typical cubic-spline interpolation.

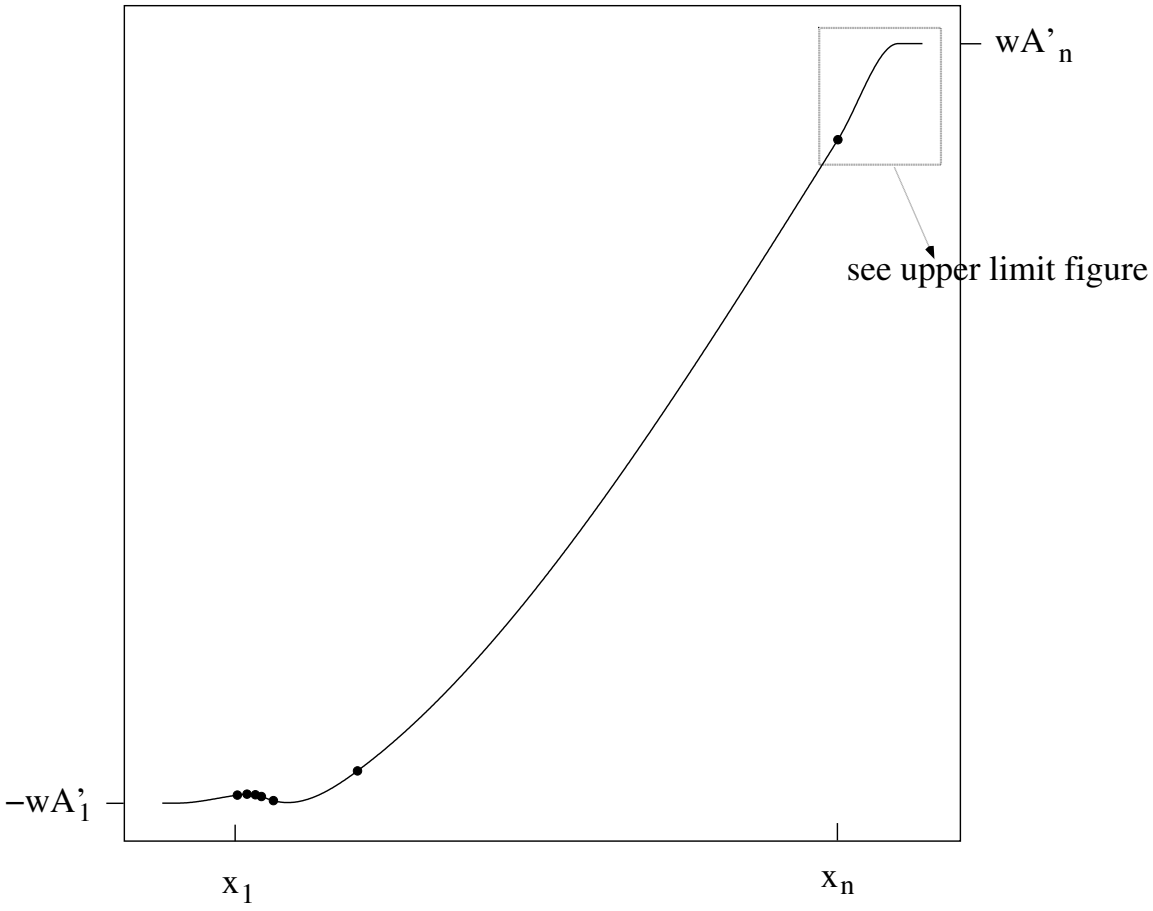


Figure F.1: Typical interpolation

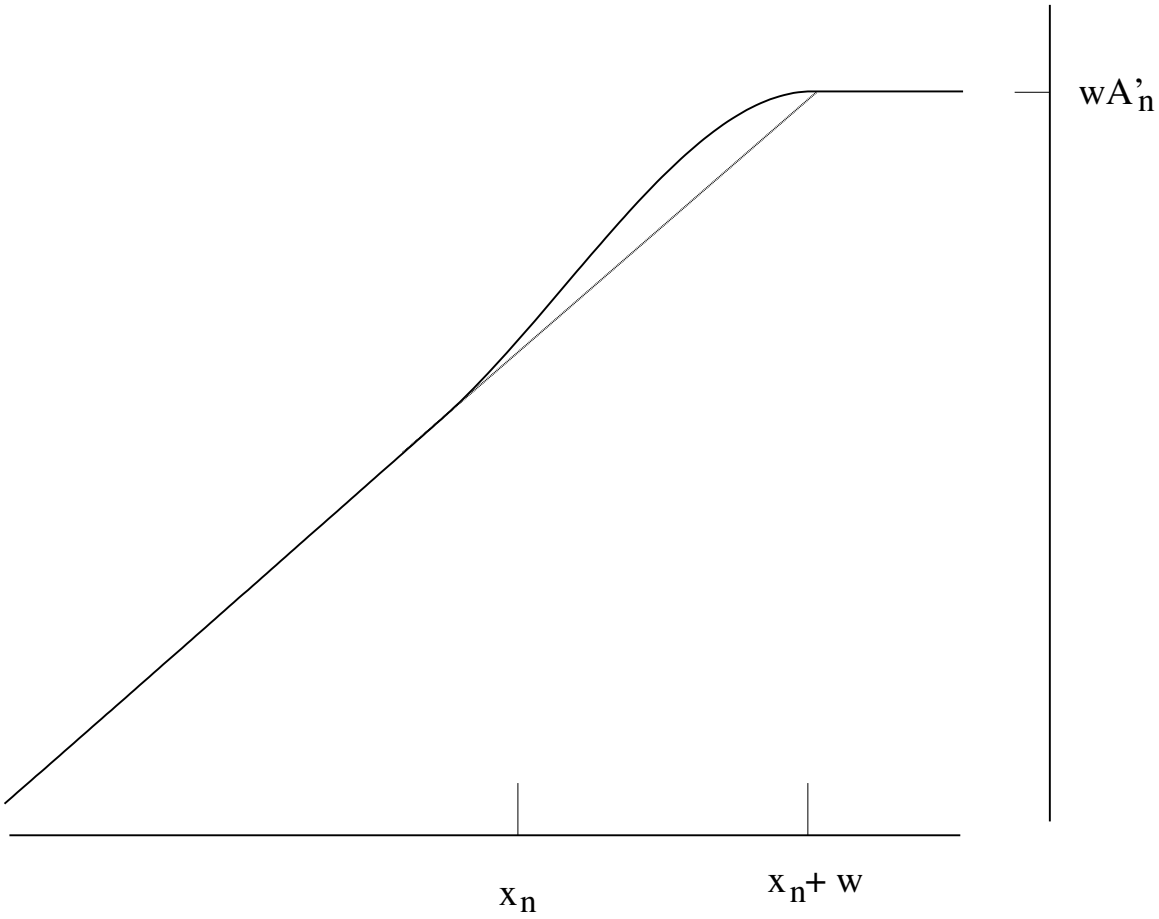


Figure F.2: Smoothed Transition at Upper Limit

Appendix G

Describing Functions

Flutter solutions in the stab module are linear in the generalized-coordinates, and in general nonlinear in all other parameters. The describing function technique [45][30] can be used to approximate flutter results for nonlinear generalized- coordinate displacements. Assuming the nonlinearity affects only one element of the stiffness matrix, that stiffness is replaced by a function of the displacement amplitude. This *equivalent stiffness* is dependent on the type of nonlinearity. For example, for a *gap* (or *freeplay* or *dead zone*), the force-displacement curve and it's describing function approximation are shown in figure G.1. The describing-function approximation to the force curve is a sinusoidal with the same frequency as the displacement; thus there is a linear relationship between the input (displacement) and the output (force) and we can use a linear flutter solution. That is the reason for choosing the describing-function to have the same form (sinusoidal) as the displacement.

The criteria for fitting the sine function to the nonlinear force curve is minimizing the least-squares error. This gives the same result as you would get if you did a Fourier transform of the force function and took the first term. A least-squares fit of sinusoidals requires evaluating an integral over one cycle. Another way of looking at this process is the nonlinear force curve is *projected* onto the describing-function sinusoidal. To project a vector onto another vector you do a dot-product or *inner-product*; for example, the projection of a vector $\boldsymbol{y}$ onto another vector $\boldsymbol{x}$ is $\frac{\boldsymbol{x}\boldsymbol{x}^t}{\boldsymbol{x}^t\boldsymbol{x}}\boldsymbol{y}$. In the time domain inner products are integrals and the projection of $f(t)$ onto $\sin\omega t$ is

$$f_\delta(t) = \frac{\omega}{\pi}\sin\omega t \int_0^{2\pi/\omega} f(t)\sin\omega t dt \tag{G.1}$$

where the inner product of $\sin\omega t$ with itself is $\frac{\pi}{\omega}$.

Continuing with the example of a gap δ in a stiffness element, the force over one cycle of sinusoidal oscillation with displacement

$$x(t) = x_0 \sin\omega t \tag{G.2}$$

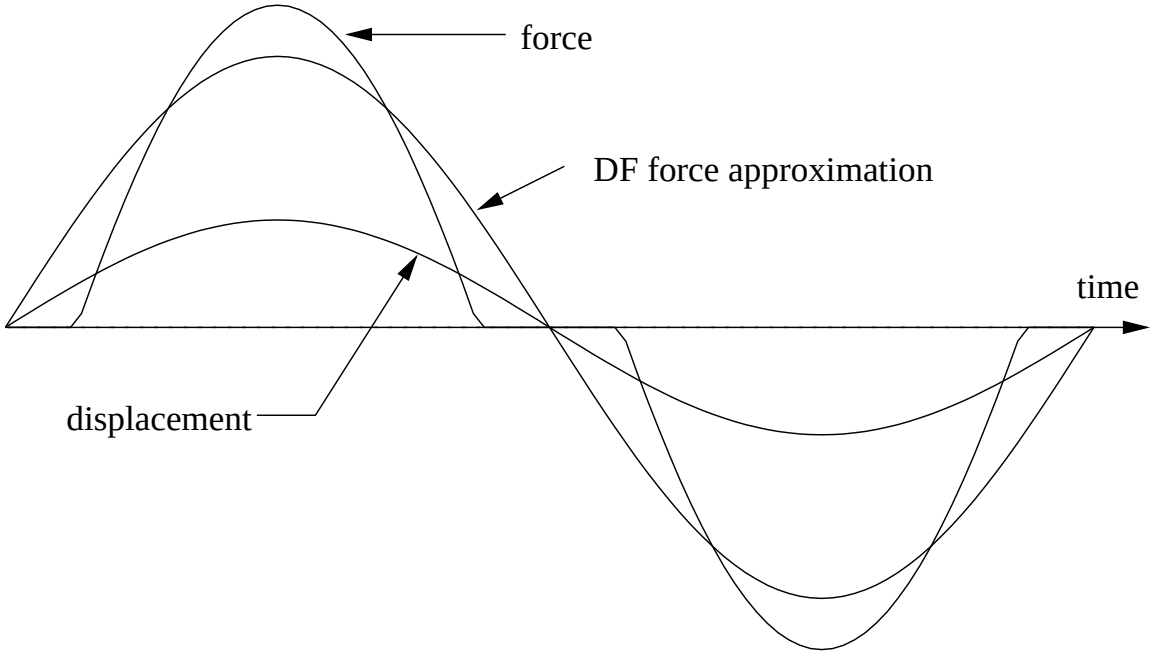


Figure G.1: Describing Function Approximation of Freeplay

is

$$f(t) = \begin{cases} 0 & 0 \le t \le t_\delta \\ k(x-\delta) & t_\delta \le t \le \frac{\pi}{\omega} - t_\delta \\ 0 & \frac{\pi}{\omega} - t_\delta \le t \le \frac{\pi}{\omega} + t_\delta \\ k(x+\delta) & \frac{\pi}{\omega} + t_\delta \le t \le 2\frac{\pi}{\omega} - t_\delta \\ 0 & 2\frac{\pi}{\omega} - t_\delta \le t \le 2\frac{\pi}{\omega} \end{cases} \tag{G.3}$$

Substituting (G.3) into (G.1), the approximate force (as illustrated in figure G.1) is

$$f_\delta(t) = \frac{k}{\pi}\left[\pi - 2\sin^{-1}\frac{\delta}{x_0} - \sin\left(2\sin^{-1}\frac{\delta}{x_0}\right)\right] x_0 sin\omega t \tag{G.4}$$

from which the equivalent stiffness is

$$k_\delta = kc \tag{G.5}$$

where

$$c = \frac{1}{\pi}\left[\pi - 2\sin^{-1}\frac{\delta}{x_0} - \sin\left(2\sin^{-1}\frac{\delta}{x_0}\right)\right] \tag{G.6}$$

is the *describing function* for a structural gap; figure G.2 shows this describing function as a function of the ratio δ/x_0.

As another example of this technique consider a mechanism with a failure: a degree-of-freedom has stiffness in one direction but not the other. In addition, the freedom has preload - a static force acts upon it, giving it an initial displacement h:

$$x(t) = h + x_0 \sin\omega t \tag{G.7}$$

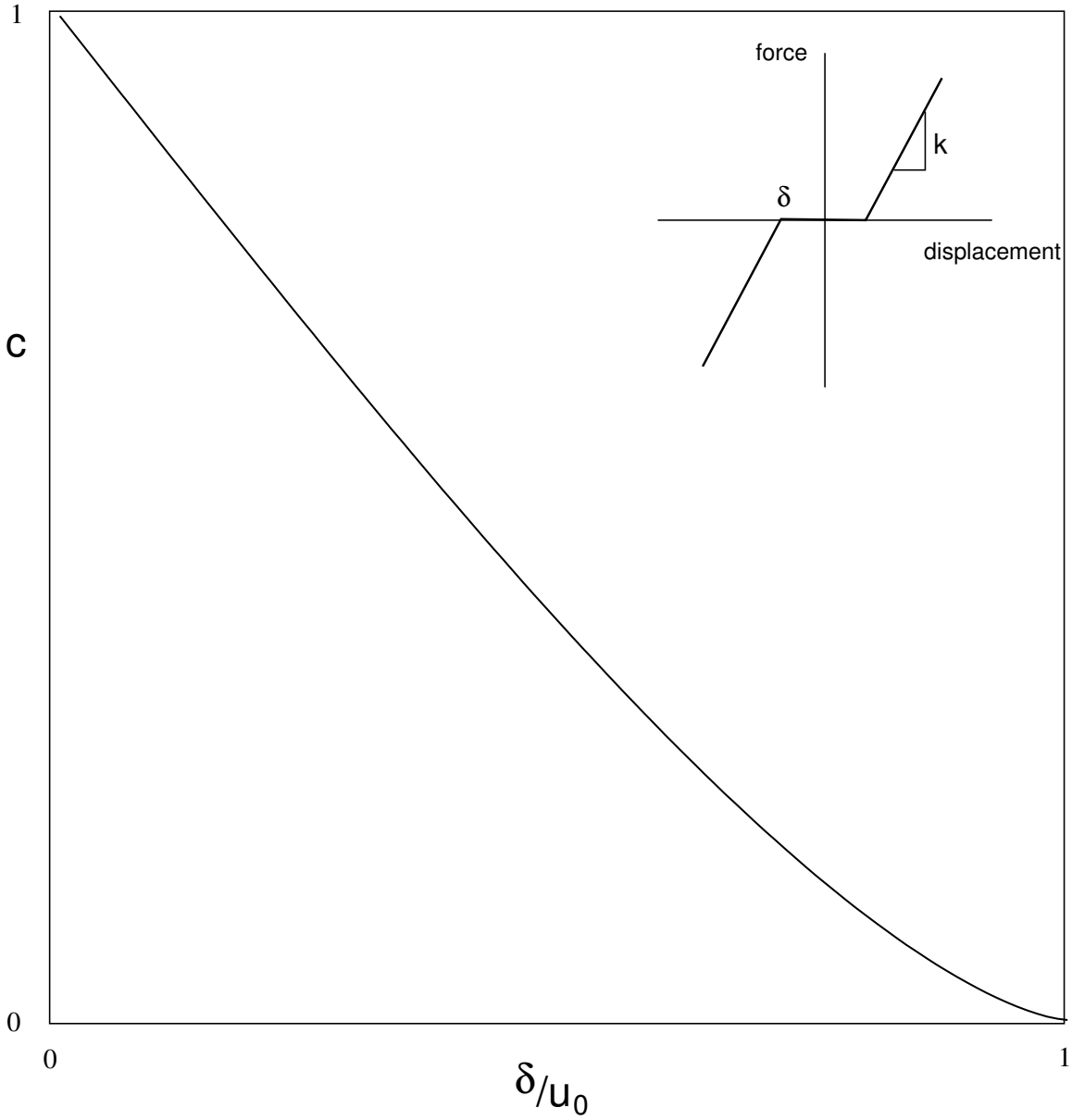


Figure G.2: Describing Function for Freeplay

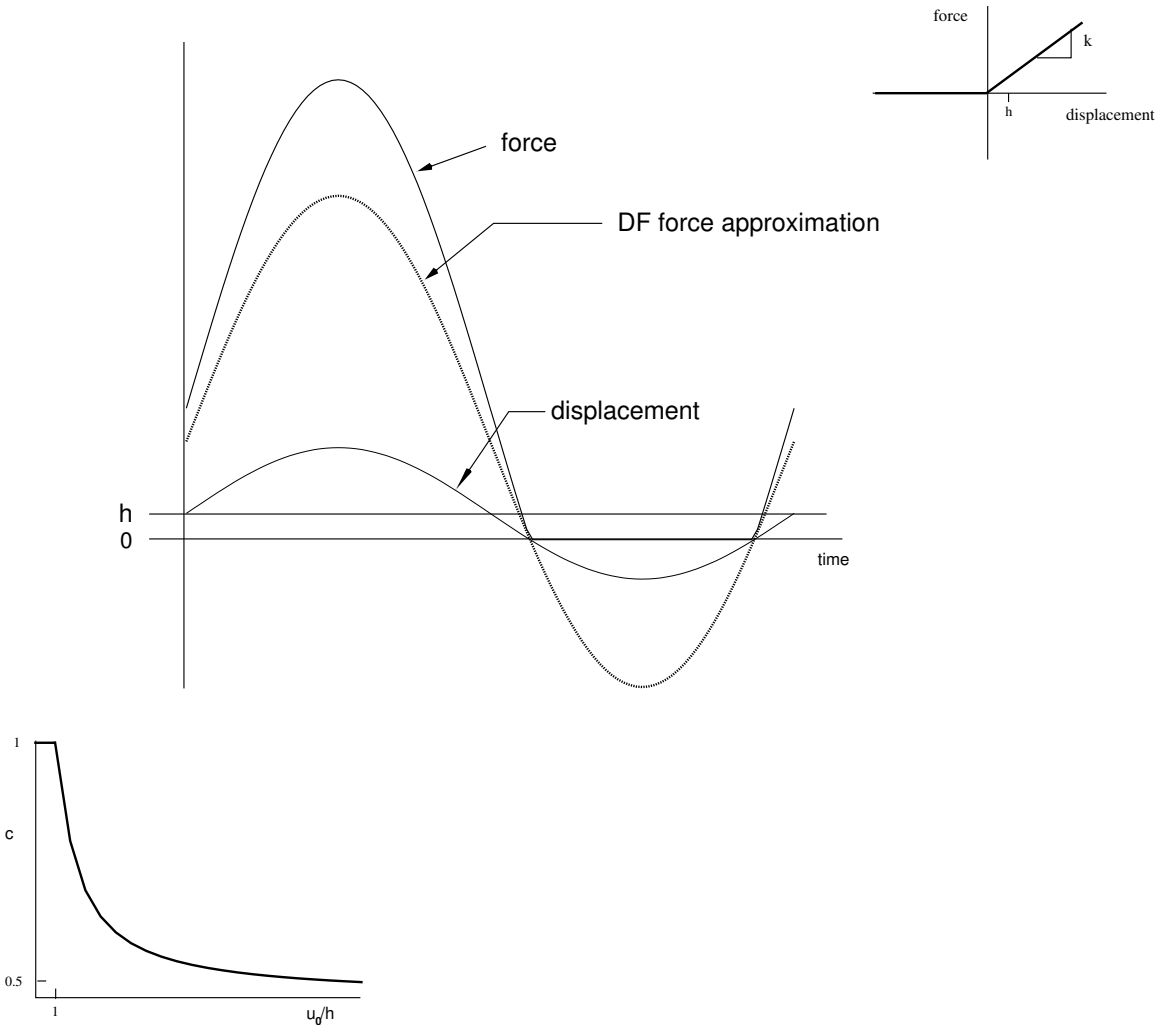


Figure G.3: Describing Function Approximation of Failed Mechanism with Preload

The describing-function is

$$c = \frac{1}{2} + \frac{1}{\pi}\left[\sin^{-1}\left(\frac{h}{x_0}\right) + \frac{h}{x_0}\sqrt{1-\left(\frac{h}{x_0}\right)^2}\right] \tag{G.8}$$

Figure G.3 illustrates this equation along with the force-displacement curve, the force, equivalent-force and displacement-time curves, and the describing-function. This describing function is available as built-in function ffwp (sect. 5.4.4).

Next consider another failed mechanism with a static displacement on the failed (zero stiffness side of the force-displacement curve (figure G.4). The describing function for this force-displacement curve is almost the same as before:

$$c = \frac{1}{2} - \frac{1}{\pi}\left[\sin^{-1}\left(\frac{h}{x_0}\right) + \frac{h}{x_0}\sqrt{1-\left(\frac{h}{x_0}\right)^2}\right] \tag{G.9}$$

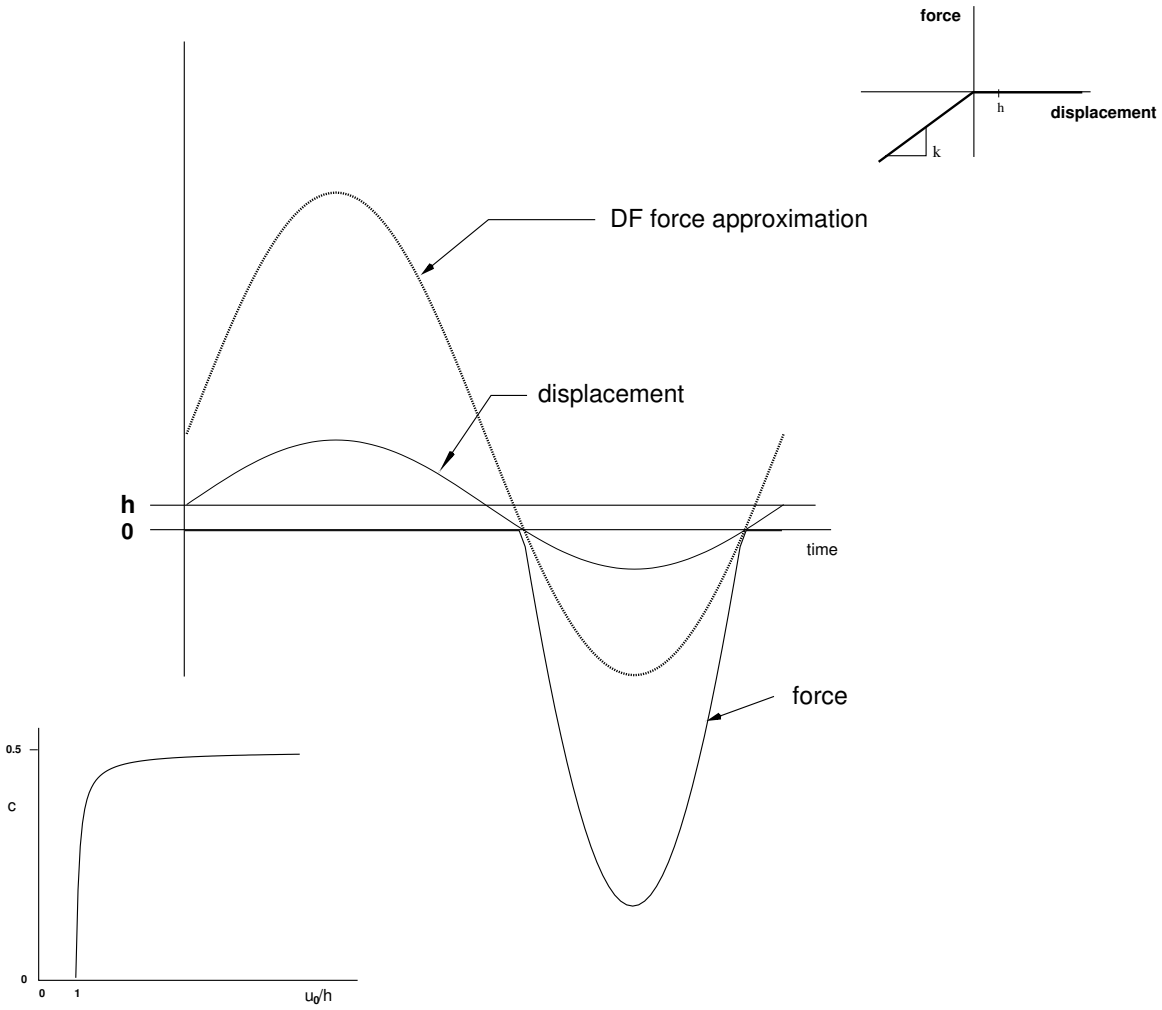


Figure G.4: Describing Function: Failed Mechanism Not Against the Stop

This describing function is available as built-in function ffnas (sect. 5.4.4).

G.1 Using Describing Functions

Describing functions have been described as a *quasi-linearization* technique because the stiffness is no longer a nonlinear function of time, but is still a function of the amplitude of oscillation. The usual technique for applying describing functions to nonlinear problems is to assume a displacement, then using the ratio of gap to displacement compute the equivalent stiffness from the describing function. Applying the equivalent stiffness to the structure and new set of displacements can be computed. This iteration is repeated until it converges.

Once we have the displacement for the gap coordinate we scale the vector of generalized coordinates to give the computed displacement at the nonlinear generalized coordinate:

$$\boldsymbol{q} \leftarrow \frac{x}{q_k}\boldsymbol{q} \tag{G.10}$$

These generalized coordinates can then be used in output transformations to get displacements at other points.

Then we create plots of flutter speed versus displacement at the gap coordinate for a specified gap size.

Appendix H

Substructuring for Dynamic Analyses

Substructures are, as the name implies, pieces of a finite-element model which are assembled to form the structure being modeled. Depending on your point of view, the equivalent term *superelement* might be preferable: a substructure is piece of a larger structure or a superelement is a collection of finite-elements. Here we use the terms interchangeably.

Substructures are used to break a large structure into smaller pieces to:

- allow the design and analysis of a structure to be split among different groups
- allow quick, easy and efficient modification of substructures, for example nacelle natural frequency variations (chap. **??**)
- make the solution more economical by reducing the size of the model and allowing modifications to be carried out on only part of the model.

Superelements may be classified as static or dynamic depending on the type of coordinate representation used. Coordinate representations are either physical or generalized degrees-of-freedom (dof). Physical dof are also referred to as *nodal* dof. Generalized dof, or *generalized coordinates*, are a set of independent quantities that represent all possible motions of a structure. Nodal dof are generalized dof but not vice-versa. Associated with each generalized dof is a *basis* function, and each generalized dof is said to be *based* on it's basis function. Generalized-coordinates are sometimes referred to as *participation factors* because they are the amount each basis function participates in the motion of the structure. A common example of generalized-coordinate basis functions is free-vibration modes.

A static substructure is simply a piece of a structure represented in nodal dof (translations and rotations). A static substructure could consist of a single finite-element; the process for assembling static substructures is identical to the assembly of finite elements.

A dynamic superelement is created from a static superelement by replacing some or all of the nodal dof with generalized dof, usually based on vibration modes, and usually reducing the number of degrees of freedom in the process. Reducing the number of degrees of freedom is often one of the primary reasons for using dynamic superelements; equally important is the ability to modify parts of the structure easily.

H.1 Static Substructuring

Static superelements are merged by creating a matrix of the correct size and inserting the matrix elements in the appropriate spot with overlapping elements accumulating. For example if superelement A has stiffness matrix

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{ii} & \boldsymbol{A}_{ib} \\ \boldsymbol{A}_{bi} & \boldsymbol{A}_{bb} \end{bmatrix} \tag{H.1}$$

where the i subscripts refer to interior dof and the b subscripts refer to boundary dof, and superelement B has

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{bb} & \boldsymbol{B}_{bi} \\ \boldsymbol{B}_{ib} & \boldsymbol{B}_{ii} \end{bmatrix} \tag{H.2}$$

then the merged substructure C will have

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}_{ii} & \boldsymbol{A}_{ib} & \boldsymbol{0} \\ \boldsymbol{A}_{bi} & \boldsymbol{A}_{bb} + \boldsymbol{B}_{bb} & \boldsymbol{B}_{bi} \\ \boldsymbol{0} & \boldsymbol{B}_{ib} & \boldsymbol{B}_{ii} \end{bmatrix} \tag{H.3}$$

A typical merged static superelement matrix is illustrated in figure H.1.

In general there is mass and stiffness coupling between the interior and boundary dof of a static substructure, hence there is also mass and stiffness coupling between merged substructures.

H.2 Dynamic Substructuring

Dynamic substructuring techniques transform some or all of the nodal coordinates to a new basis computed using both the mass and stiffness matrices, usually with the objective of reducing the size of the model. In this sense Guyan reduction (§3.1.1) is a dynamic substructuring technique. At Boeing the most commonly used techniques are *Component Modal Synthesis* (CMS) and *Branch Modes*.

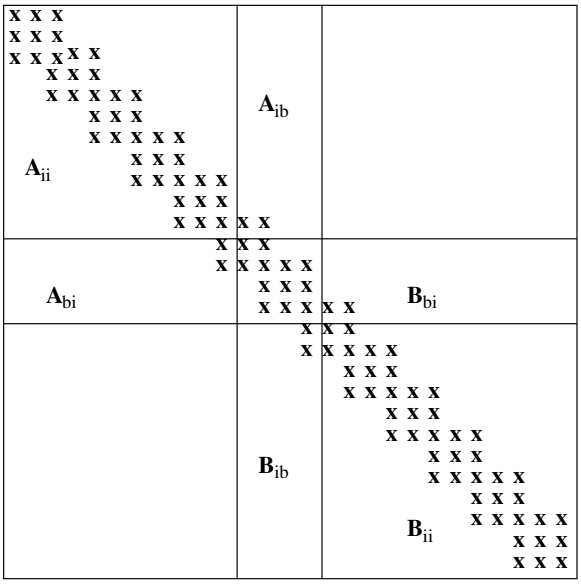


Figure H.1: Merged static superelements A and B

H.2.1 Component Mode Synthesis (CMS)

The terms Component Mode Synthesis or *Modal Synthesis* refer to a number of similar methods; among these the method of Craig and Bampton [11] is the most straightforward and is available in ATLAS, Elfini, and NASTRAN.

In the Craig and Bampton [11] approach a dynamic superelement is a superelement that has the same physical degrees-of-freedom on the boundary as a static superelement, but the interior dof are replaced with generalized coordinates based on vibration modes computed with the boundary freedoms supported. Typical mass and stiffness matrices for a dynamic superelement are shown in figure H.2.

Two characteristics of this method make it particularly attractive for many structural-dynamics analyses:

- **They can be treated like static superelements**. Because the boundary dof are left untouched, the dynamic superelement can be merged with other (static or dynamic) superelements **as though it were a static superelement**. For example, if superelement A is a static superelement, while superelement B is a dynamic superelement, the merged mass and stiffness matrices will appear as in figure H.3.
- **No stiffness coupling**. The other important characteristic of Craig and Bampton CMS is that there is no stiffness coupling between the interior dof and the boundary dof so when the substructure is merged with other substructures there is no stiffness coupling between the interior dof and other substructures. Nor is there stiffness coupling between interior dof (the interior portion of the stiffness matrix is diagonal). This property makes it easy to do parameter studies of the influence of stiffness of a substructure on, for example flutter speed.

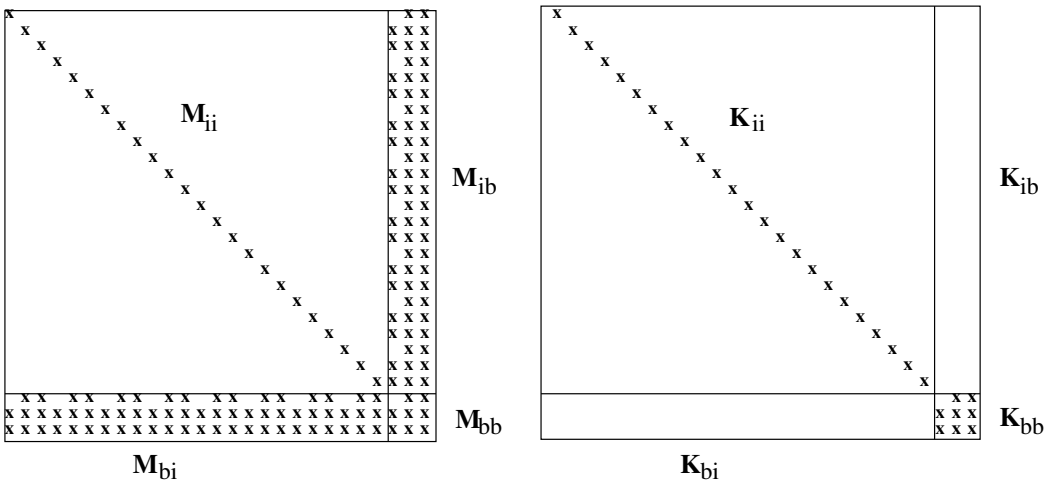


Figure H.2: Dynamic superelement mass and stiffness

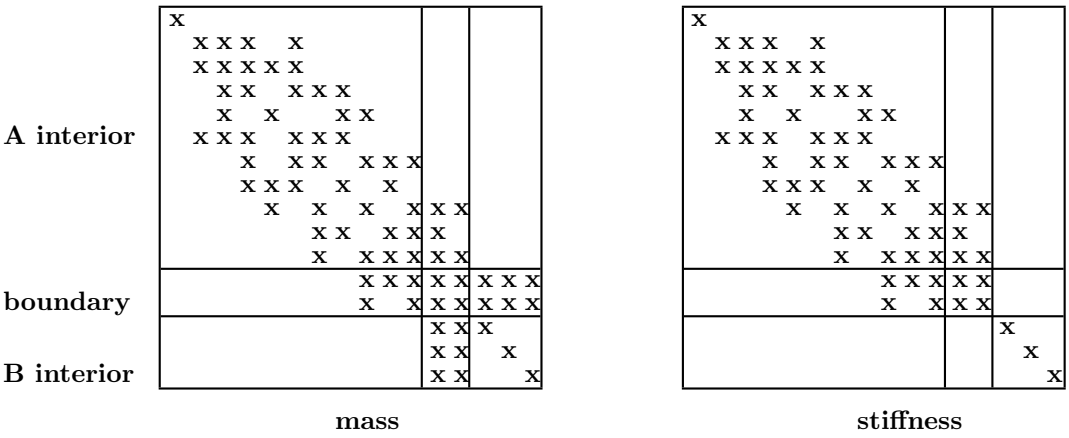


Figure H.3: Merged static (A) and dynamic (B) superelements

Once a structure comprising dynamic superelements is assembled the boundary dof are no longer of interest and it would be desirable to eliminate them to reduce the size of the problem for subsequent dynamic analyses; the Branch Modes method (§H.2.2) is a way to do this.

Mathematical Details

A dynamic superelement is formed from a static superelement by the coordinate transformation

$$\begin{aligned} \bar{\boldsymbol{K}} &= \boldsymbol{\Phi}^T \boldsymbol{K} \boldsymbol{\Phi} \\ \bar{\boldsymbol{M}} &= \boldsymbol{\Phi}^T \boldsymbol{M} \boldsymbol{\Phi} \end{aligned} \tag{H.4}$$

where $\boldsymbol{\Phi}$ is the (n, n_k) dynamic superelement modes matrix, $\boldsymbol{M}$ and $\boldsymbol{K}$ are the (n, n)

static superelement mass and stiffness matrices, and $\bar{\boldsymbol{M}}$ and $\bar{\boldsymbol{K}}$ are the (n_k, n_k) dynamic superelement generalized mass and stiffness matrices. The dynamic superelement modes matrix consists of free-vibration modes with the boundary freedoms supported and so-called *constraint modes*. Constraint modes are the displacement pattern resulting from giving each boundary freedom in turn a unit displacement while the remaining boundary freedoms are fixed. Mathematically this is equivalent to solving the linear equations

$$\boldsymbol{K}\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{K}_{ii} & \boldsymbol{K}_{ib} \\ \boldsymbol{K}_{bi} & \boldsymbol{K}_{bb} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Phi}_{ib} \\ \boldsymbol{I}_{bb} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0}_{ib} \\ \boldsymbol{F}_{bb} \end{bmatrix} \tag{H.5}$$

where the stiffness matrix $\boldsymbol{K}$ has been partitioned into interior and boundary freedoms denoted by i and b subscripts, respectively, and $\boldsymbol{F}_{bb}$ are the forces necessary to produce the unit displacements. These forces are of no interest so we simply need to solve

$$\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ib} = -\boldsymbol{K}_{ib} \tag{H.6}$$

for the constraint modes $\boldsymbol{\Phi}_{ib}$; notice that this transformation is equivalent to the the transformation used for static condensation (eqn 3.6).

Free-vibration modes are computed using the interior partitions of the mass and stiffness matrices:

$$\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ik} = \boldsymbol{M}_{ii}\boldsymbol{\Phi}_{ik}\boldsymbol{\Lambda}_{kk} \tag{H.7}$$

where $\boldsymbol{\Lambda}_{kk}$ is an (n_k, n_k) diagonal matrix of eigenvalues, and the number of computed modes $0 \leq n_k \leq n_i$. The n_k computed modes are a basis for the generalized coordinates representing the interior degrees-of-freedom.

The assembled dynamic superelement modes matrix is

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_{ik} & \boldsymbol{\Phi}_{ib} \\ \boldsymbol{0} & \boldsymbol{I}_{bb} \end{bmatrix} \tag{H.8}$$

Substituting H.5 and H.7 into H.4:

$$\begin{aligned} \bar{\boldsymbol{K}} &= \begin{bmatrix} \boldsymbol{\Phi}_{ik} & \boldsymbol{\Phi}_{ib} \\ \boldsymbol{0} & \boldsymbol{I}_{bb} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{K}_{ii} & \boldsymbol{K}_{ib} \\ \boldsymbol{K}_{bi} & \boldsymbol{K}_{bb} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Phi}_{ik} & \boldsymbol{\Phi}_{ib} \\ \boldsymbol{0} & \boldsymbol{I}_{bb} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{\Phi}_{ik}^T\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ik} & \textit{symmetric} \\ \boldsymbol{\Phi}_{ib}^T\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ik} + \boldsymbol{K}_{bi}\boldsymbol{\Phi}_{ik} & \boldsymbol{\Phi}_{ib}^T\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ib} + \boldsymbol{K}_{bi}\boldsymbol{\Phi}_{ib} + \boldsymbol{\Phi}_{ib}^T\boldsymbol{K}_{ib} + \boldsymbol{K}_{bb} \end{bmatrix} \end{aligned} \tag{H.9}$$

Using the fact that $\boldsymbol{K}_{bi} = \boldsymbol{K}_{ib}^T$ (symmetry) and the definition of constraint modes (H.6)

$$\boldsymbol{\Phi}_{ib}^T\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ik} + \boldsymbol{K}_{bi}\boldsymbol{\Phi}_{ik} = \left(\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ib} + \boldsymbol{K}_{ib}\right)^T\boldsymbol{\Phi}_{ik} = \boldsymbol{0} \tag{H.10}$$

and

$$\boldsymbol{\Phi}_{ib}^T \boldsymbol{K}_{ii} \boldsymbol{\Phi}_{ib} = -\boldsymbol{K}_{bi} \boldsymbol{\Phi}_{ib} = -\boldsymbol{\Phi}_{ib}^T \boldsymbol{K}_{ib} \tag{H.11}$$

which explains one reason why constraint modes are defined the way they are: **there is no stiffness coupling between the boundary and interior freedoms**. Furthermore

$$\bar{\boldsymbol{K}}_{kk} = \boldsymbol{\Phi}_{ik} \boldsymbol{K}_{ii} \boldsymbol{\Phi}_{ik} \tag{H.12}$$

is diagonal. Using these relations the generalized stiffness matrix $\bar{\boldsymbol{K}}$ can be written as

$$\begin{aligned} \bar{\boldsymbol{K}} &= \begin{bmatrix} \bar{\boldsymbol{K}}_{kk} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{K}_{bb} + \frac{1}{2}\left(\boldsymbol{K}_{ib}^T \boldsymbol{\Phi}_{ib} + \boldsymbol{\Phi}_{ib}^T \boldsymbol{K}_{ib}\right) \end{bmatrix} \\ &= \begin{bmatrix} \bar{\boldsymbol{K}}_{kk} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{K}_{bb} - \boldsymbol{\Phi}_{ib}^T \boldsymbol{K}_{ii}^T \boldsymbol{\Phi}_{ib} \end{bmatrix} \\ &= \begin{bmatrix} \bar{\boldsymbol{K}}_{kk} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{K}_{bb} - \boldsymbol{K}_{ib}^T \boldsymbol{K}_{ii}^{-1} \boldsymbol{K}_{ib} \end{bmatrix} \end{aligned} \tag{H.13}$$

In the last of these three alternative forms the block $\boldsymbol{K}_{bb} - \boldsymbol{K}_{ib}^T \boldsymbol{K}_{ii}^{-1} \boldsymbol{K}_{ib}$ is the static condensation (§3.1.1) of the substructure stiffness matrix to the boundary dof. The generalized mass matrix,

$$\bar{\boldsymbol{M}} = \begin{bmatrix} \boldsymbol{\Phi}_{ik}^T \boldsymbol{M}_{ii} \boldsymbol{\Phi}_{ik} & \mathit{symmetric} \\ \boldsymbol{\Phi}_{ib}^T \boldsymbol{M}_{ii} \boldsymbol{\Phi}_{ik} + \boldsymbol{M}_{bi} \boldsymbol{\Phi}_{ik} & \boldsymbol{\Phi}_{ib}^T \boldsymbol{M}_{ii} \boldsymbol{\Phi}_{ib} + \boldsymbol{M}_{bi} \boldsymbol{\Phi}_{ib} + \boldsymbol{\Phi}_{ib}^T \boldsymbol{M}_{ib} + \boldsymbol{M}_{bb} \end{bmatrix} \tag{H.14}$$

shows that there is, in general, mass coupling between the boundary and interior generalized coordinates. The lower-right block is recognized as the Guyan reduction (§3.1.1) [18] of the mass matrix to the boundary dof.

If the coordinate transformation is applied to an arbitrary vector

$$\bar{\boldsymbol{v}} = \begin{bmatrix} \bar{\boldsymbol{v}}_i \\ \bar{\boldsymbol{v}}_b \end{bmatrix} \tag{H.15}$$

where here again the k subscript refers to the "kept" cantilevered vibration modes and the b refers to the boundary dof, the resulting vector $\boldsymbol{v}$ is

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v}_i \\ \boldsymbol{v}_b \end{bmatrix} = \boldsymbol{\Phi} \bar{\boldsymbol{v}} = \begin{bmatrix} \boldsymbol{\Phi}_{ik} & \boldsymbol{\Phi}_{ib} \\ \mathbf{0} & \boldsymbol{I}_{bb} \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{v}}_k \\ \bar{\boldsymbol{v}}_b \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}_{ik} \bar{\boldsymbol{v}}_k + \boldsymbol{\Phi}_{ib} \bar{\boldsymbol{v}}_b \\ \bar{\boldsymbol{v}}_b \end{bmatrix} \tag{H.16}$$

which shows that the boundary coordinates are not affected by the transformation. In particular if $\boldsymbol{r}$ is a rigid-body vector for the static superelement, then

$$\bar{\boldsymbol{r}} = \left\{ \begin{array}{c} \mathbf{0} \\ \boldsymbol{r}_b \end{array} \right\} \tag{H.17}$$

is a rigid-body vector for the dynamic superelement, because

$$\begin{aligned} \boldsymbol{K}\boldsymbol{r} &= \left[\begin{array}{cc} \boldsymbol{K}_{ii} & \boldsymbol{K}_{ib} \\ \boldsymbol{K}_{bi} & \boldsymbol{K}_{bb} \end{array} \right] \left\{ \begin{array}{c} \boldsymbol{\Phi}_{ib}\boldsymbol{r}_b \\ \boldsymbol{r}_b \end{array} \right\} = \left\{ \begin{array}{c} \boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ib}\boldsymbol{r}_b + \boldsymbol{K}_{ib}\boldsymbol{r}_b \\ \boldsymbol{K}_{bi}\boldsymbol{\Phi}_{ib}\boldsymbol{r}_b + \boldsymbol{K}_{bb}\boldsymbol{r}_b \end{array} \right\} \\ &= \left\{ \begin{array}{c} -\boldsymbol{K}_{ib}\boldsymbol{r}_b + \boldsymbol{K}_{ib}\boldsymbol{r}_b \\ -\boldsymbol{\Phi}_{ib}^T\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ib}^T\boldsymbol{r}_b + \boldsymbol{K}_{bb}\boldsymbol{r}_b \end{array} \right\} \\ &= \left\{ \begin{array}{c} \mathbf{0} \\ \boldsymbol{K}_{bb}\boldsymbol{r}_b - \boldsymbol{\Phi}_{ib}^T\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ib}\boldsymbol{r}_b \end{array} \right\} = \mathbf{0} \end{aligned} \tag{H.18}$$

by definition, so that

$$\bar{\boldsymbol{K}}\bar{\boldsymbol{r}} = \left\{ \begin{array}{c} \mathbf{0} \\ \boldsymbol{K}_{bb}\boldsymbol{r}_b - \boldsymbol{\Phi}_{ib}^T\boldsymbol{K}_{ii}\boldsymbol{\Phi}_{ib}\boldsymbol{r}_b \end{array} \right\} = \mathbf{0} \tag{H.19}$$

also. Rigid body motion is therefore only a function of the boundary dof. This fact also follows from the fact that the boundary portions of the generalized mass and stiffness matrices are Guyan-reductions of the interior freedoms.

H.2.2 Branch Modes

The Branch Modes method [17] for substructured beam models has been used extensively at Boeing for several years. Among the reasons for its popularity is that it produces generalized mass and stiffness matrices in which certain generalized coordinates are based on clamped vibration modes of one of the substructures **and** there is no stiffness coupling between these coordinates and any other coordinates. These two conditions which, we call the *Branch Mode property*, make it easy to do parameter studies with vibration frequencies of one of the substructures, since the frequency is directly related to only one (diagonal) element of the generalized stiffness. Unlike CMS, Branch Modes does not retain boundary freedoms so the resulting models have fewer dof. Moreover, the resulting modes are often much better approximations to actual motions encountered in stability and response solutions. The disadvantage to the method is added complexity in implementing it.

For a two-substructure problem (either static or dynamic substructures - fig. H.3) the Branch Modes method can be briefly described as:

- choose one of the substructures (say A) to be the **root** substructure and the other (B) to be the **branch**

- compute vibration modes of A with B constrained to move rigidly with A
- compute vibration modes of B clamped at the A-B interface
- merge these two sets of modes and reduce the mass and stiffness using the result

Because B was constrained to move rigidly when modes of A were computed, there is no stiffness coupling between modes of A and B. Furthermore, some of the generalized coordinates are based on clamped vibration modes of B. These two conditions are the Branch Mode property.

Constraining superelement B to move rigidly with A is equivalent to lumping the mass of B onto the A-B boundary; more generally Guyan-reduce superelement B to the A-B boundary and add the reduced mass and stiffness to the A boundary. If B is a CMS dynamic superelement the boundary portions of the mass and stiffness matrices are already Guyan-reductions.

The branch modes method as presented by Gladwell [17] works provided the interface between A and B is statically determinate; otherwise it is unclear how to constrain B to move rigidly without at the same time constraining the boundary freedoms relative to each other. This requirement restricts the original method to beam models with single-node boundaries between superelements. Here we present a generalization of the original method which allows for statically indeterminate interfaces so it can be used with FEM models.

As noted earlier the branch modes procedure is straightforward when applied to substructured models with statically-determinate interfaces; with a statically-indeterminate interface it is not clear how to constrain superelement B to move rigidly with A without constraining the boundary freedoms relative to one another. This is where the use of a dynamic superelement reduction of B can be used to advantage, since the motions of the interior freedoms and boundary freedoms are separated in a natural way. Instead of constraining the interior freedoms to move rigidly relative to the boundary freedoms, we simply support (remove from the vibration analysis) the generalized coordinates which represent the motion of the interior freedoms. This is **almost** the same as constraining the interior freedoms to move rigidly relative to the boundary freedoms: more precisely, it corresponds to Guyan-reducing [18] B to the boundary freedoms between A and B. Motion of the boundary freedoms will result in non-rigid motion of the interior freedoms of B according to the definition of constraint modes. [1] This type of motion is exactly what we want to prevent unnatural constraints on the vibration modes of A.

If the merged mass and stiffness matrices are reduced to generalized coordinates based on the vibration modes of the merged structure the branch mode property is lost. What is needed is a generalized-coordinate transformation based on modes which are capable of representing the true vibration characteristics of the structure while retaining the branch mode property. In this section we show how to reduce the merged static and dynamic superelement matrices to generalized coordinates while retaining the branch mode nature of the stiffness matrix.

[1] see section H.2.1

Mathematical Details

Assuming for the moment that we have available merged mass and stiffness matrices for static superelement A and dynamic superelement B

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{ii}^{A} & \boldsymbol{K}_{ib}^{A} & \boldsymbol{0} \\ \boldsymbol{K}_{bi}^{A} & \boldsymbol{K}_{bb}^{A} + \bar{\boldsymbol{K}}_{bb}^{B} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \bar{\boldsymbol{K}}_{kk}^{B} \end{bmatrix} \tag{H.20}$$

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_{ii}^{A} & \boldsymbol{M}_{ib}^{A} & \boldsymbol{0} \\ \boldsymbol{M}_{bi}^{A} & \boldsymbol{M}_{bb}^{A} + \bar{\boldsymbol{M}}_{bb}^{B} & \bar{\boldsymbol{M}}_{bi}^{B} \\ \boldsymbol{0} & \bar{\boldsymbol{M}}_{ib}^{B} & \bar{\boldsymbol{M}}_{kk}^{B} \end{bmatrix} \tag{H.21}$$

The vibration problem for superelement A with rigid B amounts to solving

$$\left(\boldsymbol{K}_{nn}^{A+} - \omega^2 \boldsymbol{M}_{nn}^{A+}\right) \boldsymbol{\phi} = \boldsymbol{0} \tag{H.22}$$

where the subscript n represents all freedoms in A (i and b) and the + superscript refers to the additional mass and stiffness terms from B on the boundary,

$$\boldsymbol{K}_{nn}^{A+} = \begin{bmatrix} \boldsymbol{K}_{ii}^{A} & \boldsymbol{K}_{ib}^{A} \\ \boldsymbol{K}_{bi}^{A} & \boldsymbol{K}_{bb}^{A} + \bar{\boldsymbol{K}}_{bb}^{B} \end{bmatrix} \tag{H.23}$$

and

$$\boldsymbol{M}_{nn}^{A+} = \begin{bmatrix} \boldsymbol{M}_{ii}^{A} & \boldsymbol{M}_{ib}^{A} \\ \boldsymbol{M}_{bi}^{A} & \boldsymbol{M}_{bb}^{A} + \bar{\boldsymbol{M}}_{bb}^{B} \end{bmatrix} \tag{H.24}$$

That is, solve the vibration problem for superelement A with superelement B Guyan-reduced to the A-B boundary and added to the mass and stiffness matrices for A. If B is a dynamic superelement the boundary mass and stiffness terms **are** Guyan reductions of B.

including boundary terms from dynamic superelement B. As shown in appendix A, this is equivalent to solving the vibration problem for A with superelement B Guyan-reduced to the interface freedoms.

Solving eqn. H.22 for m modes

$$\boldsymbol{\Phi}_{nm}^{A+} = \begin{bmatrix} \boldsymbol{\Phi}_{im} \\ \boldsymbol{\Phi}_{bm} \end{bmatrix} \tag{H.25}$$

Reduced to this basis, the generalized mass and stiffness for A are

$$\begin{aligned}\bar{\boldsymbol{M}}_{mm}^{A+} &= \left(\boldsymbol{\Phi}_{nm}^{A+}\right)^T \boldsymbol{M}_{nn}^{A+}\boldsymbol{\Phi}_{nm}^{A+} \\ &= \boldsymbol{\Phi}_{im}^T\boldsymbol{M}_{ii}^A\boldsymbol{\Phi}_{im} + \boldsymbol{\Phi}_{im}^T\boldsymbol{M}_{ib}^A\boldsymbol{\Phi}_{bm} \\ &\quad + \boldsymbol{\Phi}_{bm}^T\boldsymbol{M}_{bi}^A\boldsymbol{\Phi}_{im} + \boldsymbol{\Phi}_{bm}^T\left(\boldsymbol{M}_{bb}^A + \boldsymbol{M}_{bb}^B\right)\boldsymbol{\Phi}_{bm}\end{aligned} \tag{H.26}$$

and

$$\begin{aligned}\bar{\boldsymbol{K}}_{mm}^{A+} &= \left(\boldsymbol{\Phi}_{nm}^{A+}\right)^T \boldsymbol{K}_{nn}^{A+}\boldsymbol{\Phi}_{nm}^{A+} \\ &= \boldsymbol{\Phi}_{im}^T\boldsymbol{M}_{ii}^A\boldsymbol{\Phi}_{im} + \boldsymbol{\Phi}_{im}^T\boldsymbol{K}_{ib}^A\boldsymbol{\Phi}_{bm} \\ &\quad + \boldsymbol{\Phi}_{bm}^T\boldsymbol{K}_{bi}^A\boldsymbol{\Phi}_{im} + \boldsymbol{\Phi}_{bm}^T\left(\boldsymbol{K}_{bb}^A + \boldsymbol{K}_{bb}^B\right)\boldsymbol{\Phi}_{bm}\end{aligned} \tag{H.27}$$

Now consider the transformation matrix

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_{nm}^{A+} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_{kk} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}_{im} & \mathbf{0} \\ \boldsymbol{\Phi}_{bm} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_{kk} \end{bmatrix} \tag{H.28}$$

which when used to reduce the merged superelement matrices results in

$$\bar{\boldsymbol{K}} = \boldsymbol{\Phi}^T\boldsymbol{K}\boldsymbol{\Phi} = \begin{bmatrix} \bar{\boldsymbol{K}}_{mm}^{A+} & \mathbf{0} \\ \mathbf{0} & \bar{\boldsymbol{K}}_{kk}^{B} \end{bmatrix} \tag{H.29}$$

and

$$\bar{\boldsymbol{M}} = \boldsymbol{\Phi}^T\boldsymbol{M}\boldsymbol{\Phi} = \begin{bmatrix} \bar{\boldsymbol{M}}_{mm}^{A+} & \bar{\boldsymbol{M}}_{mk} \\ \bar{\boldsymbol{M}}_{km} & \bar{\boldsymbol{M}}_{kk}^{B} \end{bmatrix}$$

where the mass coupling term

$$\bar{\boldsymbol{M}}_{km} = \bar{\boldsymbol{M}}_{mk}^T = \boldsymbol{M}_{kb}^B\boldsymbol{\Phi}_{bm}$$

$\bar{\boldsymbol{K}}$ and $\bar{\boldsymbol{M}}$ represent the entire structure (A and B) in a modal basis with the following characteristics:

- $\bar{\boldsymbol{M}}_{kk}^B$ and $\bar{\boldsymbol{M}}_{mm}^{A+}$ are diagonal but $\bar{\boldsymbol{M}}$ is not, due to the mass coupling term $\bar{\boldsymbol{M}}_{km}$
- $\bar{\boldsymbol{K}}$ is diagonal
- the last k generalized-coordinates are based on clamped vibration modes of B

The last two characteristics mean the branch mode property has been retained. Figure H.4 illustrates the form of the generalized mass and stiffness.

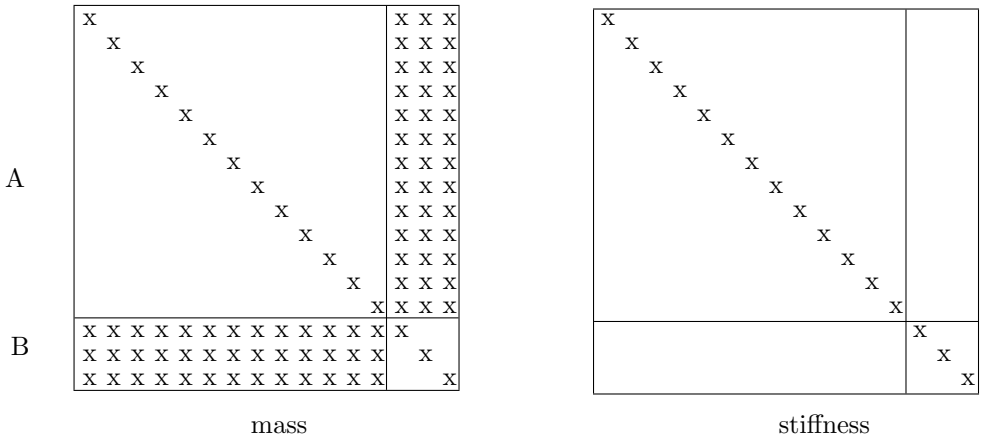


Figure H.4: Branch Mode generalized mass and stiffness

H.2.3 Example: Strut Frequency Variations

Flutter modes of an airplane are often sensitive to the fundamental frequencies of the strut/engine/nacelle, so flutter groups study the variation of flutter speed with strut frequencies. These variations have in the past been simplified by using the branch mode technique. While branch modes will not work with finite element models, the technique described above provides an analogous technique.

There are significant advantages to importing the strut/engine/nacelle from NASTRAN as a dynamic superelement:

- the correct loadpaths are maintained between the wing and strut
- the dynamics of the strut/engine/nacelle can be represented to any desired accuracy by increasing the number of interior generalized coordinates.
- much less data needs to be transmitted between groups; a 30000 dof finite-element engine model could be represented with a dynamic superelement with less than a hundred dof with more accuracy than current beam representations.
- strut frequency parameter variations can be easily and accurately performed

Appendix I

Regular Expressions

Regular expressions are a very important part of using Unix effectively. They are used in a few places in FLAPS, for example to print a catalog (10.7) of a set of FLAPS matrices.

A regular expression is a sequence of characters that describes in a very compact form, how to match a pattern of characters. Like math symbols, certain characters have very precise meaning in a regular expression. Learning the language of regular expressions is not too difficult, but it does take some study and a bit of practice. With a good understanding of regular expressions you can do some remarkable things to text files.

Unfortunately some characters have different meanings when interpreted by a pattern-matching program such as `sed` or `grep`, and a file-handling program such a the shell. For example, the asterisk (*) is interpreted by the shell to mean zero or more instances of any character. When used in a regular expression, it means "the previous character repeated zero or more times".

A regular expression is composed of three types of characters:

- **anchor characters** specify the positioning of the regular expression in a line of text
- **ordinary characters** which match themselves
- **modifiers** specify how many times the previous character is to be repeated.

I.1 Anchor Characters

Anchor characters are the caret (^) which means "beginning of the line", and the dollar sign ($) which means "end of line". So, for example,

```
^s
```

matches a line beginning with S,

```
S$
```

matches a line ending with S, and

```
^$
```

matches an empty line.

I.2 Ordinary Characters

are all characters except the so-called **metacharacters** which have special meaning in regular expressions. If you want to match a metacharacter it must be preceded (**escaped**) by a backslash (\). A dot (.) matches any character; to match a dot itself, escape the dot: '\.'. A **range** of characters can be specified by enclosing the range in square brackets ([]). A range may be simply an explicit set of characters or a hyphenated range like a-z. A character set in square brackets can be negated by starting the range with a caret (^); thus

```
[^aeiou]
```

matches any character **except** lower-case vowels.

I.3 Modifiers

include the asterisk (*), which means "repeat the preceding character any number of times (including zero)". It is the "including zero" part that often causes confusion. 'BCAG*' will match 'BCA' and 'BCAGGG'. The asterisk may follow a character range, as in

```
Q[^u][aeiou]*[a-z]
```

which matches a string consisting of Q followed by any character but u, followed by any number of lower-case vowels, followed by any lower-case letter, as in Qeen, Qay, Qayle and other misspellings.

To specify limits on the number of repetitions for a character, the syntax is '\{*lower,upper*\} '. For example the regular expression `([0-9]\{3,6\} )` matches strings of 3, 4, 5, or 6 digit numbers.

I.4 Remembered Patterns

Often it is necessary to "remember" part of a pattern, either to repeat it later in the pattern, or to use in the replacement pattern. Consider, for example the problem

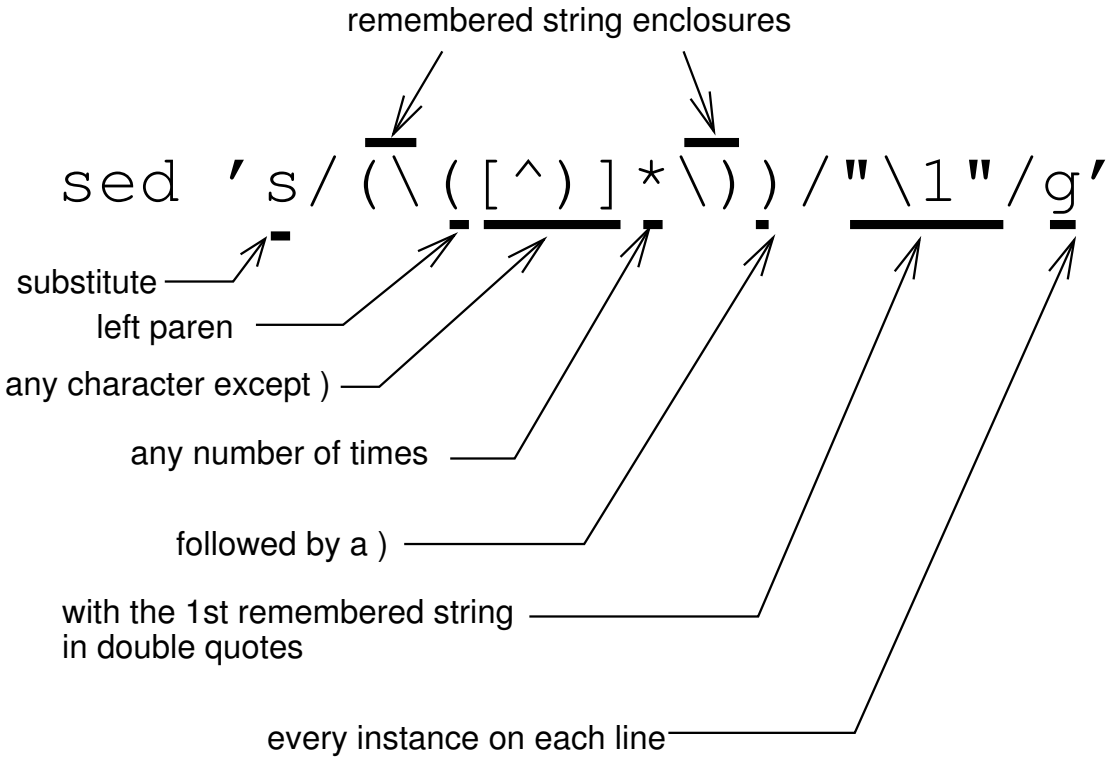


Figure I.1: A Regular Expression

of finding palindromes (words spelled the same forwards or backwards), words like eye, dud, Anna, deed, noon, or level. To find a 5-letter palindrome the pattern must "remember" the first two characters, then check that they match the 4th and 5th characters. The syntax for remembering a regular-expression (single characters being just a special case of a regular-expression) is '\(*string*\)'. The entire pattern may have up to 9 of these remembered regular-expressions. They are subsequently referred to, later in the pattern or in the replacement pattern (sed only) as \1, \2, \3, etc. For example, try typing

```
grep -i '^\([a-z]\)\([a-z]\)[a-z]\2\1$' /usr/share/dict/words
```

to see all 5-letter palindromes (words spelled the same forwards or backwards) in the system's dictionary. As an example of using a remembered string in a replacement pattern consider the following command to transform strings enclosed in parentheses to the same string enclosed in double quotes:

```
sed 's/(\([^)]*\))/"\1"/g' filename
```

Let's analyze this command piece by piece. The pattern we are looking for is a left parenthesis followed by any character except a right parenthesis, followed by a right parenthesis. It may seem like all we really need is `(.*)` (left parenthesis followed by any character any number of times followed by a right parenthesis). Unfortunately, this pattern will not work on lines containing more than one string enclosed in parentheses.

To see why, it is necessary to understand that pattern matching programs like sed and grep match the **longest possible string**, so a line like

```
(string one)  (string two)
```

is matched by '(.*)', while only the first string is matched by

```
([^)]*)
```

.

Appendix J

Interval Methods

Interval arithmetic is a technique for introducing uncertainty into calculations and for bounding solutions under the influence of uncertainty. Some algorithms can be used directly with interval arithmetic; others must be modified to get the most benefit from intervals.

J.1 Interval Arithmetic

Interval arithmetic is conceptually simple: instead of dealing with floating-point numbers, each floating-point number is represented by two values, a lower and an upper bound on the number:

$$x = [\underline{x}, \overline{x}] \tag{J.1}$$

When an arithmetic operation is performed on two intervals the result is an interval such that the operation on any number within each interval is contained in the resulting interval. For example if two intervals are added:

$$\begin{aligned} x + y &= [\underline{x}, \overline{x}] + \left[\underline{y}, \overline{y}\right] \\ &= \left[\underline{x} + \underline{y}, \overline{x} + \overline{y}\right] \end{aligned} \tag{J.2}$$

Likewise for the other basic arithmetic operations:

$$\begin{aligned} x - y &= \left[\underline{x} - \overline{y}, \overline{x} - \underline{y}\right] \\ x * y &= \left[\min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})\right] \\ x/y &= \left[\min(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}), max(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y})\right] \end{aligned} \tag{J.3}$$

Appendix K

Automatic Differentiation

A common task in scientific computing is to compute derivatives of a function. Traditional methods for doing this are numerical differentiation and symbolic differentiation. Both have significant disadvantages: numerical differentiation is prone to truncation and roundoff errors and can be expensive; symbolic differentiation requires special software to preprocess the function, such as Mathematica or Maple, and requires extra effort by the user to learn and use the software. A third alternative, *automatic differentiation* [36] is a programming technique that propagates derivatives along with the evaluation of the function. It combines the advantages of the other two techniques: it is not much more expensive than a simple evaluation of the function, does not require explicit coding of derivatives, yet it yields exact derivatives.

Traditional implementations of automatic differentiation [19] require that the function be coded in a modern programming language such as C++ or Fortran 90, using special datatypes and coding techniques. FLAPS uses a technique which allows functions written in older Fortran (Fortran 77), with no special coding on the part of the user.

K.1 Traditional Implementation

Modern programming languages such as C++ and Fortran 90 allow you to create your own datatype, called *abstract datatypes*. For purposes of illustration we define a datatype we'll call AD which will take the place of real variables. This new datatype has, in addition to a real value, a derivative. That is, each variable of this type has allocated two real values in memory: the value and the derivative which we denote here by adding a value and deriv extension to the variable name; for example

```
AD a
a.value = 1.0
a.deriv = 2.0
```

declares that variable a has a value of 1 and it's derivative is 2.

The other feature of modern programming languages that make automatic differenti-

ation possible is known as *operator overloading*. An overloaded operator like multiplication allows you to write

```
c = a*b
```

where a, b, and c are variables of type AD. The implementor of an automatic differentiation system, following the rules of differentiation, provides instructions to the compiler to translate this statement into

```
c.value = a.value*b.value
c.deriv = a.deriv*b.value + a.value*b.deriv
```

Similarly for all the other arithmetic operators (+-/). Mathematical functions such as exponentials are handled similarly, with *overloaded functions*, which have the same names as the ordinary math functions, but are called by the compiler when the argument is type AD. For example

```
AD exp(AD a) {
   b.value = exp(a.value)
   b.deriv = a.value*a.deriv*exp(a.value)
   return b
}
```

This and other overloaded math functions are also supplied by the implementor.

Now a programmer could use this automatic differentiation implementation to write programs; for example

```
main() {
   AD s
   s.value = 10.0
   s.deriv = 1.0
   a = myfunc(s)
   print "a = " a.value
   print "derivative of a wrt s = " a.deriv
}

AD myfunc(AD s) {
   return sqrt(3.0*s + s*s + s*s*s)
}
```

In this (stylized) program, s is the independent variable, so it's derivative is always 1.0. Compiling and running this little program yields

```
 a = 2.23607
 derivative of a wrt s = 1.78885
```

The only change to normal coding practice is the use of the AD datatype in place of an floating-point datatype.

K.2 Flaps Implementation

Derivatives are important in FLAPS, in particular for maintaining continuity when tracking aeroelastic modes in stab. Automatic differentiation is used in three ways in FLAPS: to compute derivatives of matrices parameterized by user-written subroutines (§6.4), to compute derivatives of arbitrary user-defined parameter equations (§5.6), and for computing first and second derivatives of ABCD control-law matrices (§6.2) with respect to the characteristic exponent s. For ABCD control-law matrices a special datatype is used which computes the first and second derivatives along with the matrix value. The derivatives are only used for computing start points for neutral-stability flutter calculations using the nonlinear eigenvalue solution technique in section **??**.

User-written subroutines, written in Fortran, must be translated into C++ in order to use automatic differentiation. This is done with a modified version of an open-source Fortran-to-C/C++ translator (f2c). After translating to C++ an automatic-differentiation datatype is substituted for real and complex variables with the active parameters as the differentiation parameters; that is, now the deriv element of an AD is an array. In most cases there are three active parameter, except for continuation optimization where there are any number greater than three.

The values of all parameters in a user-written subroutine must be obtained by calling an FLAPS function, parval which returns an AD variable with the derivatives set accordingly. For example, if the actives are vtas, freq, and growth and the user writes in the subroutine

```
freq = parval('freq')
sigma = parval('sigma')
s = cmplx(sigma, freq)
```

the parval('freq') call will return an AD object with deriv(1) set to 1.0 (since freq is an active parameter), and the parval('sigma') call will return an AD object with the value set according to the equation used in this case ($\sigma = \frac{\gamma\omega}{2}$), deriv(1) set to the partial of sigma with respect to freq ($\frac{\partial\sigma}{\partial\omega} = \frac{\gamma}{2}$), deriv(2) set to the partial of sigma with respect to growth ($\frac{\partial\sigma}{\partial\gamma} = \frac{\omega}{2}$), and deriv(3) set to the partial of sigma with respect to vtas (0). Then any arithmetic operations using these objects will propagate the derivatives with respect to the active variables, and the resulting matrix which is returned will be a matrix of AD objects with the correct derivatives.

K.2.1 Example

To compute the derivative of a matrix $\boldsymbol{A}$ with respect to a parameter w we compute the derivatives of w with respect to each active parameter a_i and sum the partial derivatives of $\boldsymbol{A}$ with respect to each active, divided by the partial of w with respect to the active:

$$\frac{\partial A}{\partial w} \leftarrow \beta\frac{\partial \boldsymbol{A}}{\partial w} + \alpha\sum\frac{\partial \boldsymbol{A}}{\partial a_i}\frac{1}{\frac{\partial w}{\partial a_i}} \tag{K.1}$$

Appendix L

Debugging

Running a debugger to find problems in programs is normally a task for developers; there are situations when users might need to know how. The most common situation is when a user-written subroutine (§6.4) is suspected of causing a program to crash. For this reason this appendix contains some basic usage instructions and some special instructions for debugging user-written subroutines.

To run an FLAPS command in the debugger, add an output statement just before the command:

```
output{debugger=ddd}
     stab { ... }
     end
```

Then when the graphical debugger ddd starts, type cont in the command window to run the program; when an error is encountered the debugger will tell you where the error occurred. If you include this information in a bug report (http://apex.ca.boeing.com/bugs) that will expedite resolution of the problem.

Appendix M

Calibrated Airspeed

M.1 The Bernoulli Equation

This appendix is due to Warren Weatherill who worked in the Boeing Flutter Research group until his retirement in 2002.

Notation: P, ρ freestream (ambient or static) conditions; P_0, ρ_0 stagnation (total) conditions; P_{sl}, ρ_{sl} are ambient sea level conditions

Integration of the steady momentum equation along a streamline results in

$$V^2/2 + \int \frac{dp}{\rho} = 0 \tag{M.1}$$

M.2 Incompressible flow

The Bernoulli equation for steady incompressible flow is simply

$$\rho V_{TAS}^2/2 = P_0 - P \tag{M.2}$$

$$V_{TAS} = \sqrt{\frac{2P(P_0/P - 1)}{\rho}} \tag{M.3}$$

Since $\rho {V_{TAS}}^2 = \rho_{sl} {V_{EAS}}^2$ then

$$V_{EAS} = \sqrt{\frac{2P(P_0/P - 1)}{\rho_{sl}}} \tag{M.4}$$

M.3 Compressible flow

The Bernoulli equation for steady compressible flow is

$$V^2/2 + \frac{\gamma}{\gamma-1}\frac{P}{\rho} = \frac{\gamma}{\gamma-1}\frac{P_0}{\rho_0} \tag{M.5}$$

$$V^2/2 = \frac{\gamma}{\gamma-1}\left(\frac{P_0}{\rho_0} - \frac{P}{\rho}\right) \tag{M.6}$$

$$V^2/2 = \frac{\gamma}{\gamma-1}\frac{P}{\rho}\left(\frac{P_0/P}{\rho_0/\rho} - 1\right) \tag{M.7}$$

For a perfect gas

$$\frac{P_0}{P} = \left(\frac{\rho_0}{\rho}\right)^{\gamma} \tag{M.8}$$

$$V^2/2 = \frac{\gamma}{\gamma-1}\frac{P}{\rho}\left[\left(\frac{P_0}{P}\right)^{\frac{\gamma-1}{\gamma}} - 1\right] \tag{M.9}$$

Noting that $\gamma P/\rho = a^2$ and $M = V^2/a^2$, there are two very useful equations that follow:

$$\left(\frac{P_0}{P}\right) = \left(1 + \frac{\gamma-1}{2}M^2\right)^{\frac{\gamma}{\gamma-1}} \tag{M.10}$$

and

$$M^2 = \left(\frac{V_{TAS}}{a}\right)^2 = \frac{2}{\gamma-1}\left(\left(\frac{P_0}{P}\right)^{\frac{\gamma-1}{\gamma}} - 1\right) \tag{M.11}$$

or

$$V_{TAS} = \sqrt{\frac{\gamma P}{\rho}\frac{2}{\gamma-1}\left(\left(\frac{P_0}{P}\right)^{\frac{\gamma-1}{\gamma}} - 1\right)} \tag{M.12}$$

and

$$V_{EAS} = \sqrt{\frac{\gamma P}{\rho_{sl}}\frac{2}{\gamma-1}\left(\left(\frac{P_0}{P}\right)^{\frac{\gamma-1}{\gamma}} - 1\right)} \tag{M.13}$$

M.4 Flaps Equation

$$V_{CAS} = 29949.18\sqrt{\left\{\frac{P}{P_{sl}}\left[(1+.2M^2)^{3.5}-1\right]+1\right\}^{0.285714}-1} \tag{M.14}$$

Re-engineering of the FLAPS equation

$$V_{CAS} = 29949.18\sqrt{\left\{\frac{P}{P_{sl}}\left[(1+\frac{\gamma-1}{2}M^2)^{\frac{\gamma}{\gamma-1}}-1\right]+1\right\}^{\frac{\gamma-1}{\gamma}}-1} \tag{M.15}$$

$$29950.4 = \sqrt{\frac{\gamma * 2116.2}{0.002378}\frac{2}{\gamma-1}} * 12.0 = \sqrt{\frac{\gamma P_{sl}}{\rho_{sl}\rho_{sl}}\frac{2}{\gamma-1}} * 12.0 \tag{M.16}$$

$$V_{CAS} = \sqrt{\frac{\gamma P_{sl}}{\rho_{sl}}\frac{2}{\gamma-1}\left\{\left[\frac{P}{P_{sl}}\left(\frac{P_0}{P}-1\right)+1\right]^{\frac{\gamma-1}{\gamma}}-1\right\}} \tag{M.17}$$

$$V_{CAS} = \sqrt{\frac{\gamma P_{sl}}{\rho_{sl}}\frac{2}{\gamma-1}\left\{\left[\frac{\Delta P}{P_{sl}}+1\right]^{\frac{\gamma-1}{\gamma}}-1\right\}} \tag{M.18}$$

Equation M.18 matches the definition of calibrated airspeed discussed in section M.5.

M.5 Definition of Calibrated Airspeed

(From McCormack (reference1), equation (2.39); also the Boeing Flight Manual)

The "calibrated airspeed," which is the speed shown on the indicated airspeed dial is defined by the following equation:

$$V_{CAS} = \sqrt{\frac{\gamma P_{sl}}{\rho_{sl}}\frac{2}{\gamma-1}\left[\left(\frac{\Delta P}{P_{sl}}+1\right)^{\frac{\gamma-1}{\gamma}}-1\right]} \tag{M.19}$$

where $\Delta P = P_0 - P$, the pressure difference that is measured by the pitot tube. Thus, the calibrated airspeed is a function of the measured pressure difference and sea level quantities. If the plane is flying at sea level, the calibrated airspeed (CAS) would be equal to the true airspeed (TAS) and also the equivalent airspeed (EAS).

$$\frac{\Delta P}{P_{sl}} = \frac{P_0 - P}{P_{sl}} = \frac{P}{P_{sl}}\left(\frac{P_0}{P_{sl}}-1\right) \tag{M.20}$$

so that

$$V_{cal} = \sqrt{\frac{\gamma P_{sl}}{\rho_{sl}} \frac{2}{\gamma - 1} \left\{ \left(\frac{P}{P_{sl}} \left[\frac{P_0}{P_{sl}} - 1 \right)^{\frac{\gamma}{\gamma - 1}} + 1 \right]^{\frac{\gamma - 1}{\gamma}} - 1 \right\}} \tag{M.21}$$

and

$$V_{cal} = \sqrt{\frac{\gamma P_{sl}}{\rho_{sl}} \frac{2}{\gamma - 1} \left\{ \left[\frac{P}{P_{sl}} \left[\left(1 + \frac{\gamma - 1}{2} M^2 \right)^{\frac{\gamma}{\gamma - 1}} - 1 \right) + 1 \right]^{\frac{\gamma - 1}{\gamma}} - 1 \right\}} \tag{M.22}$$

M.6 Dynamic Pressure

Dynamic pressure is defined as $\rho V^2/2$ For incompressible flow, the dynamic pressure is just the difference between the total pressure and the static pressure (see equation (2)). For compressible flow, the relationship is more complicated.

From equation (9) and after moving ρ from the right to the left side of the equation,

$$\rho V^2/2 = \frac{\gamma P}{\gamma - 1} \left[\left(\frac{P_0}{P} \right)^{\frac{\gamma - 1}{\gamma}} - 1 \right] \tag{M.23}$$

A more common form for dynamic pressure in compressible flow is

$$\rho V_{TAS}^2/2 = \frac{\rho}{2} \frac{\gamma P}{\rho} \frac{V_{TAS}^2}{a^2} = \gamma P M^2/2 \tag{M.24}$$

Bibliography

[1] P.W. Abrahams and B.R. Larson. *UNIX for the Impatient*. Addison-Wesley, New York, 1992.

[2] E. Albano and W.P. Rodden. A Doublet-Lattice Method for Calculating Lift Distributions on Oscillating Surfaces in Subsonic Flows. *AIAA Journal*, 7(2):279–285, 1969.

[3] R.J. Allemang. The Modal Assurance Criteria (MAC): Twenty Years of Use and Abuse. *Sound and Vibration (http://www.sandv.com/downloads/0308alle.pdf)*, pages 14–21, Aug 2003.

[4] E.L. Allgower and K. Georg. *Numerical Continuation Methods*. Springer Verlag, New York, 1990.

[5] R.L. Bisplinghoff, H. Ashley, and R.L. Halfman. *Aeroelasticity*. Addison-Wesley, Reading, MA, 1955.

[6] IEEE Standards Board. IEEE Standard for Binary Floating-Point Arithmetic. Technical Report Std 754-1985, The Institute of Electrical and Electronics Engineers, Inc, 1985.

[7] K.S.W. Champion, W.J. O'Sullivan, and S. Teweles. *U.S. Standard Atmosphere*. U.S. Govt. Printing Office,, Washington, D.C., 1962.

[8] P.C. Chen. Damping Perturbation Method for Flutter Solution: The g-Method. *AIAA Journal*, 38(9):1519–1524, Sept 2000.

[9] R.V. Churchill, J.W. Brown, and R.F. Verhey. *Complex Variables and Applications*. McGraw-Hill, 3rd edition, 1974.

[10] R.R. Craig. *Fundamentals of Structural Dynamics*. John Wiley & Sons, 2nd edition, 2006.

[11] R.R. Craig and M.C. Bampton. Coupling of Substructures for Dynamic Analyses. *AIAA Journal*, 6(7):1313–1319, July 1968.

[12] E.H. Dowell and et. al. *A Modern Course in Aeroelasticity*. Kluwer Academic Publishers, Dordrecht, The Netherlands, fourth edition, 2004.

[13] J.W. Edwards. Unsteady Aerodynamic Modeling and Active Aeroelastic Control. Technical Report SUDAAR 504, Stanfor Univ., 1977.

[14] J.W. Edwards. Applications of Laplace Transform Methods to Airfoil Motion and Stability Calculations. In *Proceedings of the 20th AIAA Structures, Structural Dynamics, and Materials Conference, Paper 79-0772*, St. Louis, MO, April 18-20 1979. AIAA.

[15] A.L. Gerth. Engineering Scientific Data (ESD) Data Format Specification. Technical Report D6-54881-200, BCAG, Seattle, WA, Jan 1991.

[16] D. Gilly. *UNIX in a Nutshell*. O'Reilly and Associates, Sebastopol, CA, 1992.

[17] G. M. L. Gladwell. Branch Mode Analysis of Vibrating Systems. *Journal of Sound and Vibration*, 1:41–59, 1964.

[18] R. J. Guyan. Reduction of Stiffness and Mass Matrices. *AIAA Journal*, 3(2):380, Feb 1965.

[19] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *C++ Toolbox for Verified Computing*. Springer-Verlag, Berlin, 1995.

[20] H.J. Hassig. An Approximate True Damping Solution of the Flutter Equation by Determinant Iteration. *AIAA Journal of Aircraft*, 8(11):885–889, Nov 1971.

[21] D.H. Hodges and G.A. Pierce. *Introduction to Structural Dynamics and Aeroelasticity*. Cambridge University Press, Cambridge, U.K., 2002.

[22] http://www.csm.ornl.gov/ kohl/MatView. *MatView Website*, 2007.

[23] http://www.mathworks.com/access/helpdesk/help/helpdesk.html. *Matlab User's Manual*, 2007.

[24] http://www.nist.gov/MatrixMarket. *Matrix Market Website*, 2007.

[25] http://www.sdrl.uc.edu/universal-file-formats-for-modal-analysis-testing 1. *Universal file formats*, 2007.

[26] J.H. Hubbard. The forced damped pendulum: chaos, complication, and control. *Amer. Math. Monthly*, 106:741–758, 1999.

[27] M.F. Hutchinson. A Fast Procedure for Calculating Minimum Cross-Validation Cubic Smoothing Splines. *ACM Transactions on Mathematical Software*, 12(2):150–153, June 1986.

[28] Boeing ISS. BCSLIB Fortran 77 Version User's Guide. Technical Report 20462-0516-R15, Boeing ISS, 1997.

[29] B.W. Kernighan and R. Pike. *The UNIX Programming Environment*. Prentice-Hall, Inc, Englewood Cliffs, NJ 07632, 1984.

[30] N. Krylov and N. Bogolyubov. *Introduction to Nonlinear Mechanics*. Princton University Press, Princton, NJ, 1947.

[31] C. Lanczos. *The Variational Principles of Mechanics*. University of Toronto Press, 1970.

[32] L. Meirovitch. *Analytical Methods in Vibration*. Prentice-Hall, Inc, New York, 1967.

[33] E.E. Meyer. Application of a New Continuation Method to Flutter Equations. In *Proceedings of the 29th AIAA Structures, Structural Dynamics, and Materials Conference Paper 88-2350*, Williamsburg, VA. (http://apex/doc/sdm88.pdf), April 18-20 1988. AIAA.

[34] W.J. Mullock. PEGASUS User's Guide. Technical Report D6-54718-600, BCAG, Seattle, WA, Jan 1991.

[35] Donal O'Regan. *Topological Degree Theory and Applications*. CRC Press, 2006.

[36] L.B. Rall. *Automatic Differentiation, Techniques and Applications*. Lecture Notes in Computer Science No. 120. Springer-Verlag, Berlin, 1981.

[37] M. Reymond. *DMAP Programmer's Guide*. MSC, Santa Ana, CA, 2004.

[38] W.C. Rheinboldt. *Numerical Analysis of Parameterized Nonlinear Equations*. John Wiley, New York, 1986.

[39] W.C. Rheinboldt. *Methods for Solving Systems of Nonlinear Equations*. SIAM, Philadelphia, second edition, 1998.

[40] W.C. Rheinboldt and J.V. Burkardt. A Locally Parameterized Continuation Process. *ACM Trans. Math. Software*, 9(2):215–235, 1983.

[41] W.P. Rodden, R.L. Harder, and E.D. Bellinger. Aeroelastic Addition to NASTRAN. Technical Report CR 3094, NASA, 1979.

[42] K.L. Roger. Airplane Math Modeling Methods for Active Control Design. In *Proceedings of the 44th Meeting of the AGARD Structures and Material Panel*, pages 4.1–4.11. AGARD CP-228, April 1977.

[43] R.M. Rosenberg. *"Analytical Dynamics of Discrete Systems"*. Plenum Press, New York, 1980.

[44] Bill Rosenblatt. *"Learning the Korn Shell"*. O'Reilly and Associates, Sebastopol, CA, 1993.

[45] D. Siljak. *Nonlinear Systems*. John Wiley and Sons, Inc., New York, 1969.

[46] R. Srinivasan. XDR: External Data Representation Standard. Technical Report 1832, The Internet Engineering Task Force (IETF), 1995.

[47] Neal Stephenson. *"In the Beginning... Was the Command Line"*. Avon Books, New York, 1999.

[48] W.R. Stevens. *Unix Network Programming Vol 1*. Prentice-Hall, Inc., New York, 1998.

[49] W.R. Stevens. *Unix Network Programming Vol 2*. Prentice-Hall, Inc., New York, 1999.

[50] W.T. Thomson. *Theory of Vibration with Applications*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

[51] M.J. Turner, R.W. Clough, H.C. Martin, and L.C. Topp. Stiffness and Deflection Analysis of Complex Structures. *Journal of the Aeronautical Sciences*, 23:805–824, 1956.

Index

D

E

Q

R

S

Y