NASA-CR-205687

Final Report - Proposal for β-test of GIFCORCODE

NASA Ames Project Number NAG 21025

by William S. Levine, Project Director

July 31, 1997

Introduction: This year's effort produced very significant progress in the development of the software package heretofore known as GIFCORCODE. One important change has been in the name. The package is now named CONDUIT for CONtrol Designer's Unified InTerface. There have also been some more significant changes in the way CONDUIT is used. These changes caused some modifications in the work accomplished. Both the original goals for the year and the modifications will be described in the next section of this report.

The major goal for this year was to bring CONDUIT to β-test. This has been accomplished. The software package is in β-test at Bell Helicopters and has been since September 1996. This and the other achievements during the past year are described in the third section of this report. Some discussion of the scaling issue is also included here. This is in answer to a question that arose at one of the CONDUIT briefings.

The report concludes with a brief set of suggestions for further work. An appendix describing the issues involved in dynamic linking is also included.

Original Goals: The primary goal for the year, as explained in the original proposal, was to bring CONDUIT to β-test. Eleven tasks that needed to be completed in order to accomplish this goal were also planned for the year.

The original eleven tasks were as follows:

Task 1: Implement a zero iterations feature.

Task 2: Change the simulations, constraints, and criteria to be more than 90

Task 3: Complete the User's Manual.

Task 4: Fully implement the Help Menus

Task 5: Combine the present three rotorcraft models into one model that can be used to evaluate all the specifications.

Task 6: Implement the frequency and time response plotting feature

Task 7: Make it possible for the designer to view multiple design parameter/constraint windows simultaneously.

Task 8: Make it possible for the designer to view the ADS-33 display in the ways that he or she finds most effective.

Task 9: Make it possible to run two copies of MATLAB simultaneously.

Task 10: Respond immediately to requests from the b-sites for help, bug notices, changes, and additional features.

Task 11: Prepare a one hour talk about CONDUIT and present it at the major rotorcraft manufacturers.

One of the first tasks to be completed was Task 11, the preparation of a one hour talk describing the software and its use. This talk was presented, as planned, at McDonnell-Douglas on August 20. 1996 and at Bell/Textron on August 21, 1996. During the discussions following these talks it was learned that neither company could use CONDUIT as long as it was restricted to Sun Computers. The helicopter companies use SGI machines.

This led to the most significant of the changes in the planned tasks. The task of porting CONDUIT to SGI computers was added to the list and made the highest priority.

Tasks 3 and 4 were also changed substantially in response to changes in the overall design of CONDUIT. The original feasibility study showed that CONSOL-OPTCAD could be very useful in the design of rotorcraft control systems. It also showed that the user of CONSOL-OPTCAD had to be very knowledgeable about control design, programming is several languages, and optimization. When Tasks 3 and 4 were proposed we thought the use of pull-down menus would greatly alleviate the need for much of the programming knowledge. As work progressed it became clear that CONDUIT could be made even easier to use by including a library of specifications. The result was to deemphasize temporarily the work on Tasks 3 and 4 and to increase substantially the planned scope of CONDUIT.

Results: The major goal for the year was achieved. CONDUIT is in β-test at Bell/Textron and has been since well before the conclusion of the year. As described earlier, a prerequisite for Bell's use of CONDUIT was that it be ported from Sun machines to SGI machines.

This involved solving a difficult problem. CONSOL-OPTCAD, the computing engine at the heart of CONDUIT, uses dynamic linking. It is difficult to ransfer dynamic linking from one type of computer to another. We were able to accomplish this task expeditiously. Appendix A contains a brief account of the reasons for dynamic linking and what was involved in the transfer.

A considerable amount of additional work was done in support of the further development of CONDUIT. This includes bug fixes, improvements to subroutines within CONDUIT, and the addition of capabilities to CONDUIT. All of this work is significant and time consuming but not worthwhile enumerating in this report.

A question was raised at one of the CONDUIT talks about scaling. The question was whether nonlinear specifications could be arbitrarily scaled within the min max framework that is the foundation for CONDUIT. The answer is yes. This is shown by the two simple MATLAB programs that follow.

```
% qu.m
% This is a little plotter of a quadratic
%WSL 2/7/97

a1=1; a2=4;
b1=-18; b2=-10;
c1=100; c2=20;
x=[0:.1:10];

min1=-b1/(2*a1)
min2=-b2/(2*a2)

y=polyval([a1 b1 c1],x);
z=polyval([a2,b2,c2],x);
plot(x,y,x,z)
title('Two quadratic functions')
xlabel('x-axis');ylabel('value of quadratics')
```

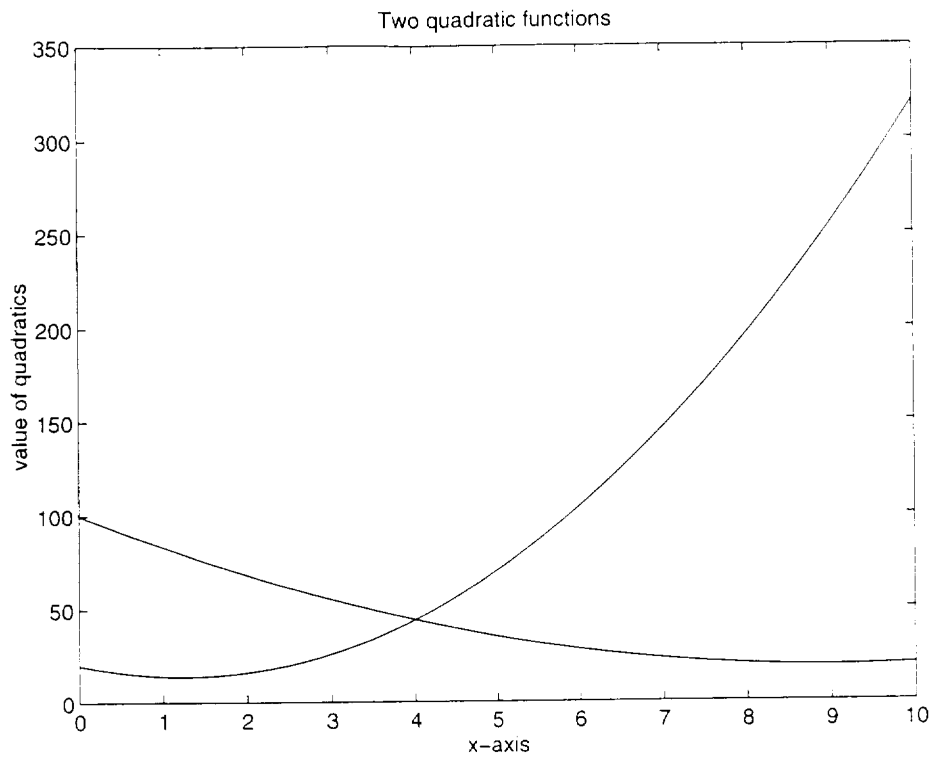

Figure 1 - Illustrating the intersection of two quadratics.

The code generates Figure 1. Notice that Figure 1 shows the intersection of two quadratic curves. The minimum of the maximums occurs at x=4. It is obvious that changing the relative y-axis scaling of the two quadratics would move the intersection anywhere on the x-axis although there might a second intersection somewhere. In contrast, the secnod piece of MATLAB code evaluates the minimum of a linear combination of the two quadratics. As illustrated in Figure 2, this minimum can only occur in the narrow range between the minimums of the individual functions.

```
% op.m
% This is a little example of linear combination of quadratic
% objectives for Mark Tischler
% WSL
% 2/6/97

% First criterion is al*x^2 + bl*x + cl
% Second criterion is a2*x^2 + b2*x = c2

% Minimizing value of x=-(bl+k*b2)/2*(al + k*a2)

clg
k=[0:.01:5];
w=-(bl*ones(size(k))+k*b2)./(2*(al*ones(size (k))+k*a2));
plot(k,w)
title('Optimizing value of x versus scaling factor')
xlabel('scaling factor'); ylabel('optmal value of x');
```

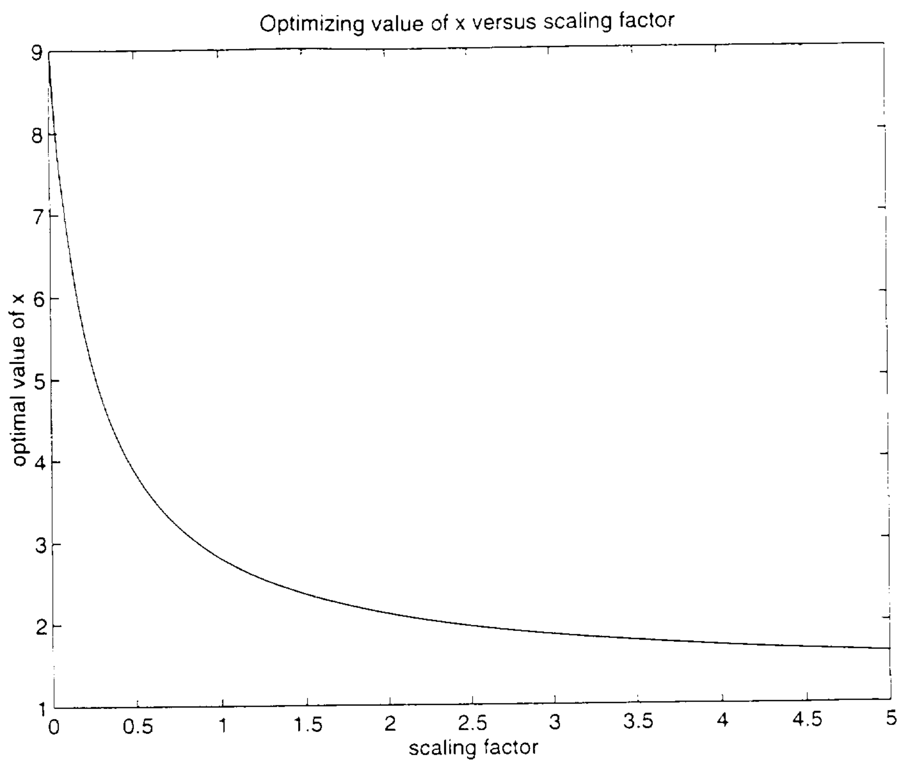

Figure 2 - The optimizing value of x versus scale factor.

Conclusions: The CONDUIT project was advanced considerably during this past year. The major objective, bringing it to β-test, was accomplished on schedule. Most of the preliminary objectives on the critical path to this major objective were changed somewhat because of changes in some aspects of CONDUIT's operation. These changes resulted from experience in using CONDUIT and from conversations with potential users. The current version, as of mid-January 1997 is greatly enhanced from the version at the beginning of the year. This was second most important objective for the year.

The current status of CONDUIT is well-documented in the paper

Conduit–A New Multidisciplinary Integration Environment
for Flight Control Development
by Mark B. Tischler, Jason D. Colbourne, Mark R. Morel, Daniel J. Biezad, William S. Levine, and Veronica Moldoveanu

which will be presented at the AIAA Guidance, Navigation, and Control Conference, August 11-13, 1997, New Orleans, Louisiana. The paper will also appear in the proceedings of that meeting.

Appendix

Implementing Dynamic Linking in CONSOL-OPTCAD

Irving Hsu

Dynamic Linking

In a program that spans several source files, a function in one file often refers to one or more symbols (e.g., variables and function names) from another file. When a program is compiled into an executable binary, resolution of such external references is performed during the **linking** stage. In this stage the compiler determines the actual memory locations of these symbols, and replaces each reference with the corresponding memory location.

Normally linking is done **statically**; that is, all symbolic references have been completely resolved by the time a binary is loaded into memory for execution. For different design projects the problem modules would be different and, if different simulators are used, each simulator would require a different interface with the solve module. Without dynamic linking, the user would need to produce an executable binary for each unique combination of design project and simulator. This clearly is not a good solution.

With dynamic linking, a more elegant alternative is possible. When the solve module is invoked, it is passed the name of the problem module and the simulator to be used. The solve module then calls upon the operating system's dynamic linking facilities to load and link in the appropriate problem module and simulator interface, and the end effect is the same as if all three modules had been statically linked.

Using Dynamic Linking

Most Unix systems provide dynamic linking services through the following calls:

```
#include    <dlfcn.h>

void*dlopen(char*path,int mode)
void*dlsym(void*handle,char*symbol);
```

dlopen() provides access to the object in path, returning a descriptor that can be used for later references to the object in calls to *dlsym()*. If path was not in the address space prior to the call to *dlopen()*, then it will be placed in the address space. When an object is brought into the address space, it may contain references to symbols whose addresses are not known until the object is loaded. These references must be relocated before the symbols can be accessed. The mode parameter governs when these relocations take place and can have the following values:

- **RTLD_NOW** all relocations take place immediately when object is loaded
- **RTLD_LAZY** relocation of a function takes place when it is first referenced

If either of these values is **OR**'d with **RTLD_GLOBAL**, the symbols contained in the object will be visible to other objects that are *dlopen'd*. The fact that the symbols from one dynamically-loaded object can be made visible to another is critical: the problem module contains references to functions defined by the simulator interface. Since both are dynamically loaded, symbols defined in the simulator interface must be made visible to the problem module for the references to be properly resolved.

IRIX 5.3 and SunOS 5.5 implementations of dynamic linking honor the **RTLD_GLOBAL** flag; SunOS 4.1.* does not. Therefore, dynamic linking with *dlopen()* and *dlsym()* will not work under SunOS 4.1.*.

dlsym() is used to determine the address binding of symbol in the object identified by handle.

With these two routines, implementing dynamic linking becomes straightforward. The solve module passes each of the object modules supplied on the command line, as well as the simulator interface selected, if any, to *dlopen()*. It also loads the problem module into its address space with *dlopen()*. In addition, using the handle returned by *dlopen()*, it calls upon *dlsym()* to locate the entry point to the problem module (the function spec_). Symbolic resolutions, if any, are taken care of automatically by the dynamic linking facilities.

Only shared objects may be dynamically linked in this manner. Thus, the convert module must be modified to produce shared objects. For *IRIX* 5.3, the compiler command is:

```
cc -ansi -shared -w -o <object>.so <object>.c
```

and for SunOS 5.5, the appropriate command is:

```
cc -G -o <object>.c
```